# IMPLEMENTATION OF THE LOCALLY COMPETITIVE ALGORITHM ON A FIELD PROGRAMMABLE ANALOG ARRAY

A Thesis
Presented to
The Academic Faculty

By

Aurèle Balavoine

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2009

# IMPLEMENTATION OF THE LOCALLY COMPETITIVE ALGORITHM ON A FIELD PROGRAMMABLE ANALOG ARRAY

Approved by:

Dr. Paul E. Hasler, Advisor
*Advisor, School of Electrical and Computer Engineering*
*Georgia Institute of Technology*

Dr. Christopher J. Rozell
*School of Electrical and Computer Engineering*
*Georgia Institute of Technology*

Dr. David V. Anderson
*School of Electrical and Computer Engineering*
*Georgia Institute of Technology*

Date Approved: November 2009

*Si tu vas devant toi pour aller devant toi,*

*C'est bien; l'homme se meut, et c'est là son emploi;*

*C'est en errant ainsi, c'est en jetant la sonde*

*Qu'Euler trouve une loi, que Colomb trouve un monde.*

*Victor Hugo*

# DEDICATION

*A ma petite soeur, Eline*

# ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Hasler, for his patience and constant support since I entered his lab. Thank you to my committee jury, Dr. Rozell and Dr. Anderson, for their input and advice all along the way. I would like to thank my lab colleagues, from whom I have learned so much, and in particular Sam Shapero who worked closely with me during this project. Of course, I would like to thank my family who permitted me to live this adventure. Thank you for supporting me wherever I went. Finally, thank you to Chris who has been present thoughout the most cheerful as well as the most difficult moments.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# SUMMARY

The goal of this thesis is to present the first attempt at realizing in analog circuits a recent optimization algorithm: the Locally Competitive Algorithm (LCA). This algorithm finds the solution to an optimization problem which is defined as the minimization of a mean-square error (MSE) constraint associated to a L1 constraint. This problem, also called Sparse Approximation, arises in a wide range of applications, such as signal processing areas related to Image Denoising, Sparse Coding or Compressed Sensing theories. The LCA is defined by a set of ordinary differential equations, which governs the dynamics of internal state variables. This set of differential equations can be realized by a neural network type of architecture composed of several nodes working in parallel, and more specifically can be classified as a Hopfield-type of neural network. In addition to the nodes, the system is composed of a threshold operator and feedbacks between the nodes. This particular architecture makes it amenable to analog circuits. An analog implementation would present several benefits over a digital approach. In particular, it may provide a much faster solution method with lower power consumption and better scaling properties.

This thesis proposes an analog system operating on sub-threshold currents as a solution. Experimental results of the circuit's components obtained on a Field Programmable Analog Array (FPAA) will be presented. While industrial fabrication is prohibitively expensive and time-consuming, an FPAA provides a reconfigurable analog platform to implement and test designs quickly and cost efficiently. Combining the power efficiency of analog circuits and the advanced mathematical theory of the LCA can possibly lead to a powerful tool with potential applications in many different fields, such as medical image processing.

# CHAPTER 1

# INTRODUCTION

In the field of Digital Signal Processing, a constant concern is to store data with as few coefficients as possible. With this goal in mind, several techniques have been developed to compress data. One of them consists of finding a sparse representation of the signal in a carefully chosen basis. A signal is said to be sparse if it has few non-zero coefficients. Wavelets and curvelets are good examples of research efforts to find sparse representations of smooth images. Exploiting the sparsity characteristic of signals was proven to be a very efficient approach in a wide range of applications, such as denoising, restoration and efficient data acquisition. The sparse approximation optimization problem finds the approximation of an input $s \in \mathbb{R}^N$ on a dictionary $\Phi$ composed of $M$ dictionary elements (generally $M \gg N$: the dictionary $\Phi$ is said to be overcomplete) with as few non-zero coefficients $a \in \mathbb{R}^M$ as possible and can be stated as follows:

$$\min_a \left( \frac{1}{2} \|s - \Phi a\|_2^2 + \lambda \|a\|_0 \right) \tag{1}$$

The first term in (1) represents the MSE between the input signal $s$ and its approximation $a$ on the basis $\Phi$. The second term represents what is known as the L0-norm of $a$, which is equal to the number of non-zero entries in $a$. Trying to minimize this term means trying to make the solution as sparse as possible. The factor $\lambda$ is a user-defined parameter which creates a tradoff between the two constraints.

The presence of the L0-norm in the equation makes the problem non-convex, and thus NP-hard. Several methods have been developed to find an approximate solution to this problem, such as Basis-Pursuit De-Noising (BPDN)[1], Gradient Descent, and Interior Point Methods. These methods all present several limitations. They are generally computationally expensive and do not allow for a parallel implementation. A classic gradient approach tries to solve a modified version of the optimization problem described in

(1), where the L0-norm has been replaced by the L1-norm. The principal challenge for these methods comes from the non-smooth nature of this new formulation. Recently developed, the LCA proposes a modified gradient descent approach to solve sparse approximation problems. Its Neural-Network type of architecture, composed of nodes connected together by inhibitive feedbacks, allows for parallel computing and is implementable in analog circuits. An analog approach would present several advantages, such as low-power consumption, a possibility to perform operations in parallel and good scaling properties. Figure 1 shows Genes law [2], which represents the expected advancement rate of digital performances in terms of power consumption per Million of Multiply Accumulate Cycles a Second (MMACS) (Figure from [3]), as well as the power consumed by an analog implementation of the same computation. The power consumed by the analog system is four orders of magnitude less than its digital counterpart. For these reasons, an analog approach of the sparse approximation problem should lead to a system with a lower time constant, lower power consumption, and which scales better as the size of the problem increases.

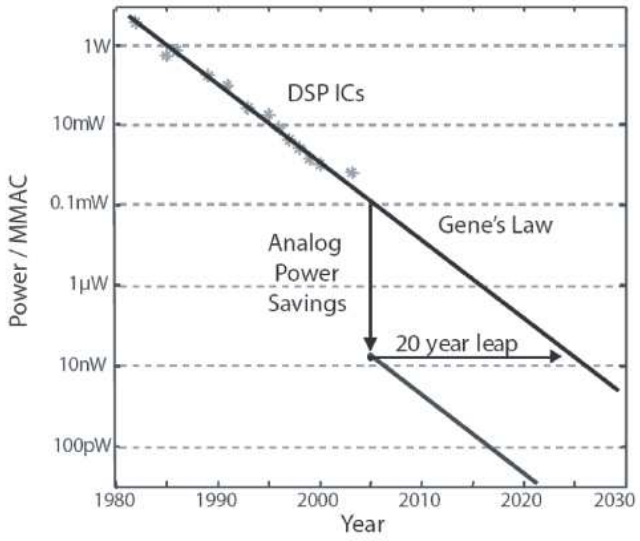


**Figure 1. Gene's law from [3]. Comparison of the power consumption of an analog and a digital system. The analog power efficiency results in a 20-year leap compared to the digital approach.**

In the first chapter, the optimization problem will be described in greater detail as well as the Locally Competitive Algorithm, along with some examples of where sparse approximation arises. Chapter 3 provides a description of the Field-Programmable Analog Array (FPAA) used to test the designed circuit, as well as the associated software used to carry out this research. Chapter 4 describes in detail the proposed circuit architecture and presents the different blocks that compose it. Chapter 5 presents the experimental results obtained on the FPAA for each block. Finally, Chapter 6 describes the problems encountered during this research and future work. Chapter 7 concludes this work.

# CHAPTER 2

# LOCALLY COMPETITIVE ALGORITHM

Sparse approximation is an important optimization problem which arises in a wide range of fields related to signal and image processing. In the first subsection, some useful mathematical notions will be briefly reviewed. The formulation of the optimization problem will then be given. The Basis-Pursuit De-noising method to solve this problem will be described. In the second subsection, some examples of where this optimization problem appears will be presented. In the last subsection, the LCA algorithm, which solves the sparse approximation problem, will be presented in details.

## 2.1 Description of the Optimization Problem

Before looking at the mathematical formulation of the problem, it is useful to do a brief review on the norms called Lp-norms.

### 2.1.1 Lp-norms

On a space where a norm can be defined ( $\mathbb{R}^N$, for example ) the set of norms, parametrized by a scalar $p \in \mathbb{R}\backslash 0$, called Lp-norms, can be defined as follows:

$$\text{for } x \in \mathbb{R}^N, \|x\|_p = \left( \sum_{k=1}^{N} x_k^p \right)^{1/p}$$

In the particular case where $p = 0$, the previous definition does not define a norm (it does not satisfy the triangular inequality) but by extension, the L0-norm is defined by:

$$\text{for } x \in \mathbb{R}^N, \|x\|_0 = \sum_{k=1}^{N} I(x_k)$$

Where $I(.)$ is the indicator function: $I(x) = \begin{cases} 1, \text{if } x \neq 0 \\ 0, \text{if } x = 0 \end{cases}$

In words, the L0-norm counts the number of non-zero entries in the signal $x$. This quality makes it a measure of sparsity.

To visualize the different Lp-norms, Figure 2 shows the balls of radius 1 for $p = 0, \frac{1}{2}, 1$ and 2, that is the location of the points with Lp-norm equal to 1: $\left\{x,\ \|x\|_p = 1\right\}$. It can be seen on this graph that: for $p < 1$, the unit balls define the contour of a non-convex space, whereas for $p \geq 1$ they define a convex space. This property is essential to understand the difficulty presented by the sparse approximation problem.

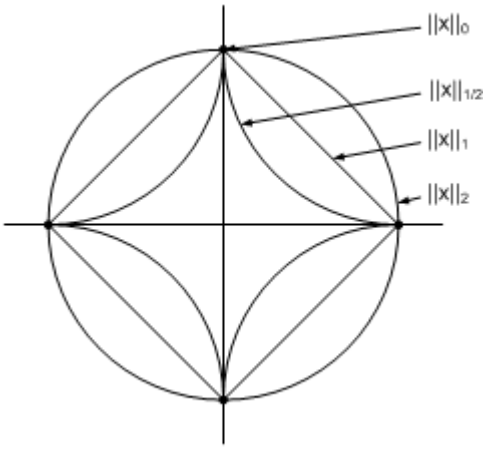


**Figure 2. The balls of radius 1 represented for different Lp-norms: $p = 0, 1/2, 2$. For $p < 1$ the space defined in non-convex. For $p \geq 1$, the space defined is convex.**

### 2.1.2 Formulation of the Problem

Sparse approximation is the desire to approximate a signal $s \in \mathbb{R}^N$ with a library $\Phi$ with as few non-zero coefficients as possible. The matrix $\Phi$ is composed of a set of $M$ dictionary elements $\Phi = [\phi_1, \ldots, \phi_M]$. Generally, $M$ is much greater than $N$, and the dictionary $\Phi$ is said to be overcomplete. The solution $a \in \mathbb{R}^N$ to the sparse approximation problem can be found by minimizing the following cost function:

$$\min_a \|s - \Phi a\|_2 \text{ such that } \|a\|_0 \leq \epsilon \tag{2}$$

The solution $a$ is the closest sparse signal, in terms of the Mean-Square Error (first term in (2)), to the input $s$ onto the dictionary $\Phi$. The sparsity constraint is introduced by the L0-norm (second term in (2)). As described previously, the L0-norm makes the optimization problem non-convex, and thus NP-hard [4]. The next section will present a classical method to approach the solution of this optimization problem.

### 2.1.3 Basis Pursuit De-Noising

The major issue in (2) is the non-convexity induced by the L0-norm. The Basis Pursuit approach consists of relaxing this constraint by using the L1-norm instead. The L1-norm is used because it produces a convex optimization problem while still inducing the necessary sparsity constraint [1]. In the absence of noise, the Basis Pursuit formulation is:

$$\Phi a = s \text{ such that } \|a\|_1 \leq \epsilon$$

In the presence of noise, the Basis Pursuit De-noising formulation [1] is used instead:

$$\min_a \|s - \Phi a\|_2 \text{ such that } \|a\|_1 \leq \epsilon$$

This can be turned into an unconstrained optimization problem by using the Lagrangian:

$$\min_a \left( \frac{1}{2} \|s - \Phi a\|_2^2 + \lambda \|a\|_1 \right) \tag{3}$$

In (3), the optimization problem is now convex. Consequently, it is possible to use classical tools for convex optimization to solve it. It was shown that the solution to this problem is exactly the solution to the original optimization problem in (2) if the signal is sparse enough [5].

Figure 3 illustrates how Basis Pursuit works. The constraint $s = \sum a_k \phi_k$ forms an hyperplane, represented by a line in two dimensions. The diamonds represent the location of points with the same L1-norm. If the L1-norm of the solution increases, the signal will lie

on a diamond of bigger diameter. The Basis Pursuit method consists of finding the intersection between the hyperplane and the diamond with smallest diameter. Graphically, this can be interpreted as increasing the diameter of the diamond until it touches the hyperplane. The intersection point corresponds to the solution to (3).

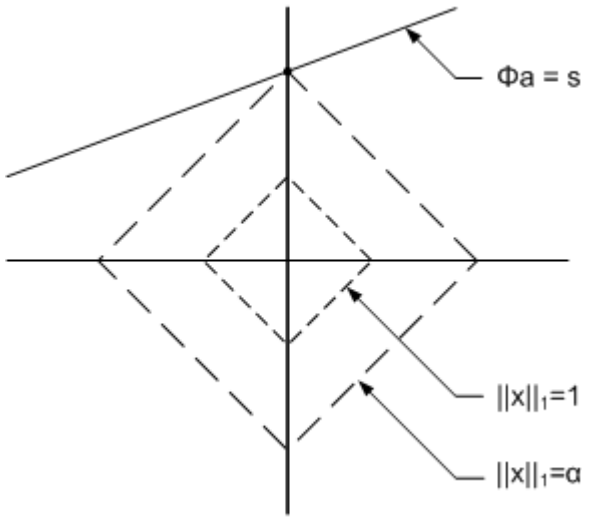


**Figure 3. Basis Pursuit Principle: The intersection of the Hyperplane with the L1 ball of smaller radius corresponds to the solution to (3). Under certain condition, this solution is also the exact solution to (2).**

## 2.2 Applications

In this subsection, three examples where this optimization problem can arise will be described. The first is approximation of signals on an overcomplete dictionary, the second is Compressed Sensing, and the third is image denoising.

### 2.2.1 Sparse Approximation

When trying to find the approximation of a signal $s \in \mathbb{R}^N$ onto a basis $\Phi$, in which all the dictionary elements are linearly independent, the inverse of $\Phi$ or its pseudo-inverse $\Phi^*$ can be directly used to get to the solution. The explicit solution is $a = \Phi^* s$.

However, if $M > N$, the dictionary defined by $\Phi$ is said to be overcomplete, and some of the dictionary elements are necessarily linearly dependent. As a consequence, there

may exist an infinite number of solutions to the problem $s \approx \sum a_k \phi_k$. In order to limit the problem to have a unique solution, it is necessary to impose a constraint in terms of a cost function on the solution.

In sparse approximation [6], the "best" solution is chosen to be the sparsest one, that is, the solution with as few non-zero coefficients as possible. This problem naturally leads to choosing the L0-norm as the constraint on the solution. The "best" approximation is the solution to the problem:

$$\min_a ||\Phi a - s||_2 \text{ such that } ||a||_0 \leq \epsilon,$$

which is the optimization problem defined in equation (2).

Having a sparse representation of a signal presents several advantages, especially for storage, transmission and computational complexity.

### 2.2.2 Inverse Problem in Compressed Sensing

An example where inverse problems arise is in the field of Compressed Sensing [7]. In this context, one tries to take efficient measurements of the signal of interest $s$ in order to compress it from the acquisition. The goal is to acquire as few coefficients as possible while still getting enough information to recover the signal accurately. The measurements are not direct samples of the signal $s$, but linear combinations of the samples of $s$: $x = \Psi s$. The theory of Compressed Sensing says that a good matrix $\Psi$ to acquire as few coefficients as necessary to recover the signal is a random matrix; that is to say, a matrix whose coefficients are drawn from a random process, such as Gaussian or Bayesian random variables. To acquire the signal, it suffices to project $s$ onto the matrix $\Psi$: $x = \Psi s$. To recover the signal $s$, the steps are slightly more complicated. The hypothesis consists of assuming the existence of a domain defined by $\Phi$ in which the signal $s$ is sparse. To recover the signal $s$, it is necessary to solve the following equation:

$$\min_a ||\Phi a - s||_2 \text{ such that } ||a||_0 \leq \epsilon$$

Or, since $x$ is known and not $s$:

$$\min_a ||\Psi\Phi a - x||_2 \text{ such that } ||a||_0 \leq \epsilon$$

Which is the same optimization problem as equation (2) with a different matrix.

### 2.2.3 Image Denoising

In the context of image denoising, a signal $s$ is corrupted with noise and the goal is to find a denoised representation of this signal. If the signal is sparse in a certain domain defined by $\Phi$, it can be represented by only a few non-zero coefficients. However, the noise will certainly be spread out on the entire spectrum on this same domain. By solving the optimization problem:

$$\min_a ||s - \Phi a||_2 \text{ such that } ||a||_0 \leq \epsilon,$$

only the coefficients which carry the most information on $s$ will be recovered, while small coefficients, which correspond to the noise, will have been set to zero. The result is a denoised version of the original image [8].

From these three examples, it is obvious that the problem that this thesis addresses is important and can arise in many different contexts.

## 2.3 LCA

The Locally Competitive Algorithm was developed in 2008 by Dr. Rozell et al [9]. It is aimed to solve the optimization problem presented in (3). It relies on the competition and inhibition between several nodes, which evolve in parallel. Its particular architecture makes it equivalent to a stable, convergent Hopfield Neural Network [10], and makes it possibly realizable in analog circuits.

### 2.3.1 ODE

The LCA is a continuous time algorithm which can be described by a set of nonlinear ordinary differential equations (ODE) which act on internal state variables $u_m(t)$. The internal states vary until they reach an equilibrium. This equilibrium corresponds to the solution of the optimization problem in (3).

The following points define some variables that appear in the formulation of the algorithm.

- $s \in \mathbb{R}^N$ is the $N$-dimensional input signal. It is assumed that the signal is sparse onto the dictionary $\Phi$.

- $\Phi$ is a $N$-by-$M$ matrix whose columns are the dictionary elements $\phi_m \in \mathbb{R}^N, m = 1, \ldots, M$: $\Phi = [\phi_1, \ldots, \phi_M]$. These dictionary elements can be called nodes, neurons or atoms.

- $u_m(t)$ for $m = 1, \ldots, M$ are functions representing the time-varying internal states of the system at time $t$. They are contained in a vector: $u(t) = [u_1(t), \ldots, u_M(t)]^t$.

- $b_m$ for $m = 1, \ldots, M$ are the driving inputs. They reflect how well the signal matches the different nodes. The closer a signal is to a node, the bigger the corresponding driving input value is: $b_m = \langle \phi_m, s \rangle$. The driving inputs are stored in the matrix: $b = [b_1, \ldots, b_M]^t = \Phi^t s$.

- As in most neural networks, a nonlinearity is introduced before the output stage in the form of a thresholding function $T_\lambda$. This function guarantees that small internal states that do not add much information to the approximation are kept to zero, while internal states which are significantly large are "active" and contribute to the output signal.

- Consequently, the vector containing the active coefficients is defined as: $a(t) = [a_1(t), \ldots, a_M(t)]^t = T_\lambda(u(t))$.

- In addition, to enforce that two internal state variables which carry the same information on the signal will not be active simultaneously after the algorithm has converged, competition between the nodes is introduced in the form of feedbacks. The strength of the feedback depends on the level of activity of the node (the more active the node, the stronger the inhibition is) and also on the match between the two competing nodes (the closer they are, the stronger the inhibition). To account for these two characteristics, the inhibition factor is proportional to $a_m G_{m,n}$, where $G_{m,n} = \langle \phi_m, \phi_n \rangle$ is the inner product between the two dictionary elements. This can be written in a matrix of the form: $G = \Phi^t \Phi - I$.

For one of the internal state variables, the nonlinear ordinary differential equation ruling the node dynamics is:

$$\begin{aligned} \dot{u}_m(t) &= \tfrac{1}{\tau}\left[b_m - u_m(t) - \sum_{n \neq m} G_{m,n} a_n(t)\right] \\ a_m(t) &= T_\lambda(u_m(t)) \end{aligned}$$

This becomes for the entire system in a matrix formulation:

$$\begin{aligned} \dot{u}(t) &= \tfrac{1}{\tau}\left[b - u(t) - (\Phi^t \Phi - I)\, a(t)\right] \\ a(t) &= T_\lambda(u(t)) \end{aligned} \tag{4}$$

Figure 4 shows a schematic of the different variables that intervene in this equation and of the different interactions between the nodes in the algorithm.

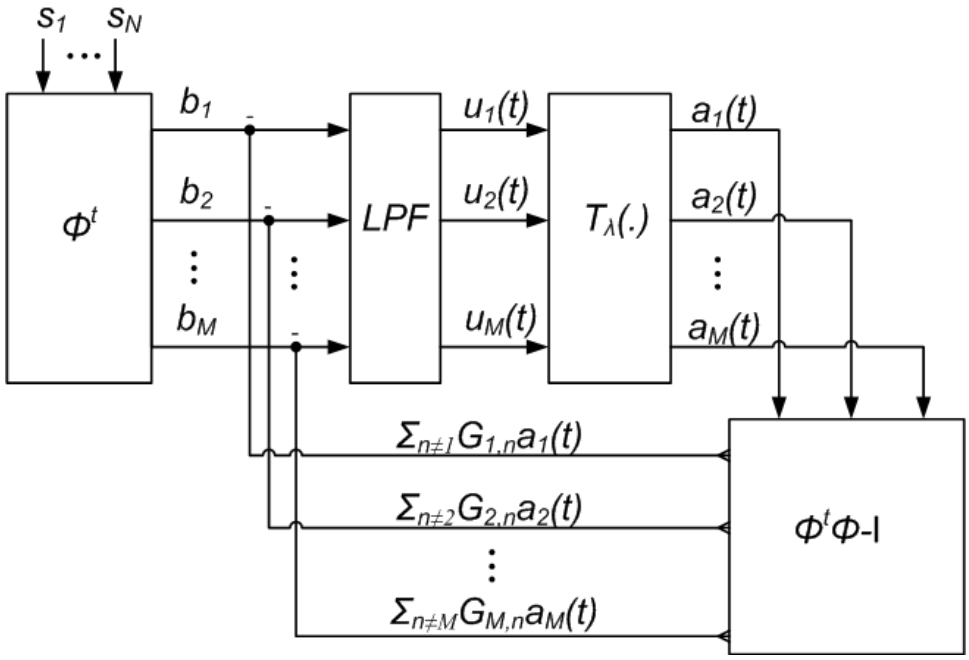


**Figure 4. Functional Block Diagram of the LCA. All the different functions and variables intervening in the algorithm are represented.**

### 2.3.2 Threshold function

The ODE described in (4) solves the sparse approximation problem presented in section 2.1.3. As mentioned previously, different constraints will induce different measures of sparsity. For a general sparsity-inducing cost penalty $C(\cdot)$, the LCA descends the corresponding energy function with respect to $C(\cdot)$ [9]:

$$E(t) = \frac{1}{2} \|s - \Phi a\|_2^2 + \lambda \sum_m C(a_m(t))$$

Rozell and al. showed that the link between the cost function $C(\cdot)$ and the threshold function $T_\lambda$ is:

$$\lambda \frac{dC(a_m)}{da_m} = u_m - a_m = u_m - T_\lambda(u_m)$$

Different threshold functions can be used to solve the optimization problem. The threshold function used in this study is the soft-threshold function [8].

$$T_\lambda(u(t)) = \begin{cases} max(u(t) - \lambda, 0), & u(t) > 0 \\ min(u(t) + \lambda, 0), & u(t) < 0 \end{cases} \tag{5}$$

The soft-threshold is plotted in Figure 5 (a) (Figure from [9]). The corresponding cost function, shown on (b) of the same figure, corresponds as expected to the L1-norm:

$$C(a_m) = |a_m|$$

This function assures that strong units are active and will suppress other units by forcing them to zero. This part of the system is essential since it is the one enforcing sparsity.

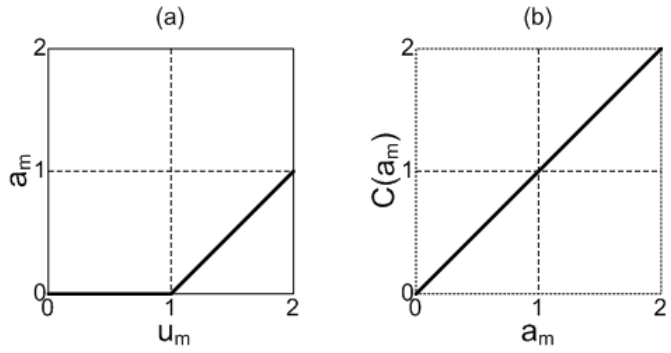


**Figure 5. Soft-Threshold Operator from [9]. (a): positive half of the soft-threshold function $T_\lambda$ (b): corresponding sparsity cost function $C = |\cdot|$**

The inputs below the threshold are set to zero, while the outputs corresponding to inputs above the threshold are linear with the inputs. The value of the threshold allows the balancing between how close the solution is in terms of the L2-norm to the real solution with how sparse it is.

The use of the soft-threshold function in the LCA formulation leads to another method of solving the optimization problem (3).

# CHAPTER 3

# FIELD PROGRAMMABLE ANALOG ARRAY

FPAAs, or Field-Programmable Analog arrays, are integrated devices that can be reprogrammed several times to achieve different analog circuits. The main advantage of using a reprogrammable platform is to allow a circuit designer to implement, test and modify its circuit as many times as necessary before sending it to fabrication. On the contrary, in a classical design process, one creates a circuit, simulates it using digital simulation tools, such as Spice, and sends it to fabrication, which can be a very long and expensive process. Once the circuit is back from fabrication, it can finally be tested on hardware. If after testing any modification needs to be done, it is necessary to go through all the previous steps again (design, simulation, fabrication and testing). After several of these iterations, the procedure can end up being very costly and time-consuming. FPAAs provide a solution to this problem by allowing the circuit designer to implement and test the circuit several times on a real piece of hardware without additional fabrication cost and in the range of an hour.

Several FPAAs, known as Reconfigurable Analog Signal Processors (RASP) have been developed in the CADSP group in the past several years [11]. The main characteristic of these types of FPAAs is the use of floating-gate transistors in the switch matrix [12]. These elements present several advantages, which will be described in the first subsection. In the second subsection, the general architecture of the RASP family of chips developed in the lab will be described. Finally, the last subsection will present different software, which have also been developed in the CADSP group to facilitate the usage of the RASP chips from design to testing.

## 3.1 Floating-Gate Elements

The main components in the RASP family of FPAA are the floating-gate transistors [13]. These transistors have their gate completely surrounded by silicon dioxide, which isolates

it from the rest of the device and allows electrons to be stored on the gate. They present several advantages because of their small size and their ability to fulfill a wide range of functions. Among those functions, they can serve as non-volatile memory elements, switches, multipliers, current generators to create bias currents or cancel offsets, etc.

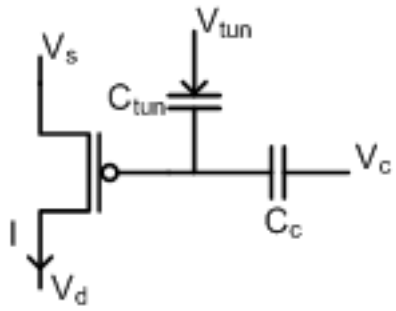


**Figure 6. Schematic of a floating gate transistor**

Figure 6 shows the schematic of a pFet floating-gate. When the device is in subthreshold operation, the current-voltage relationship is defined as:

$$I = I_s \exp\left(\frac{V_s - \kappa V_{fg}}{U_T}\right) \exp\left(\frac{V_D}{V_A}\right)$$

where $I_s$ is a current, characteristic of the device, $\kappa = \frac{C_{ox}}{C_{ox}+C_{dep}}$ where $C_{ox}$ is the oxide capacitance and $C_{dep}$ the depletion capacitance, $U_T = \frac{kT}{q}$ is the thermal voltage, $V_A$ is the Early voltage, and $V_{fg}$ is the voltage on the floating gate, whose characteristic is given in saturation by the relationship:

$$V_{fg} = V_g \frac{C_c}{C_T} + V_{offset}$$

where $C_T = C_c + C_{tun}$ is the total capacitance at the floating gate and $V_{offset}$ is determined by the charge on the floating gate. More precisely, $V_{offset} = Q/C_T$, where $Q$ is the charge stored on the floating gate.

The strength of this device lies in the possibility of precisely modifying the charge stored on the floating gate, and consequently the voltage $V_{offset}$. This programmable offset

makes this device useful in a wide range of applications. Two procedures exist to modify the charge on the floating-gate. To remove charges, Fowler-Nordheim tunneling is used, and hot-electron injection is used to add electrons to the floating gate [14].

### 3.1.1 Tunneling

Fowler-Nordheim electron tunneling is used to remove charges from the floating gate. Generally, the size of the oxide insulator is enough to prevent electrons to pass through the barrier (see Fig. 7 (a)) (Figure from [15]). To allow charges to cross the barrier, a large voltage is applied across the tunneling capacitor, which generates an electrical field across it. This leads to a decrease of the thickness of the electric barrier which allows electrons to cross (see Fig. 7 (b)). This results in a decrease in the number of electrons on the floating gate, which contributes to increasing the charge on the floating-gate and thus increasing $V_{offset}$.

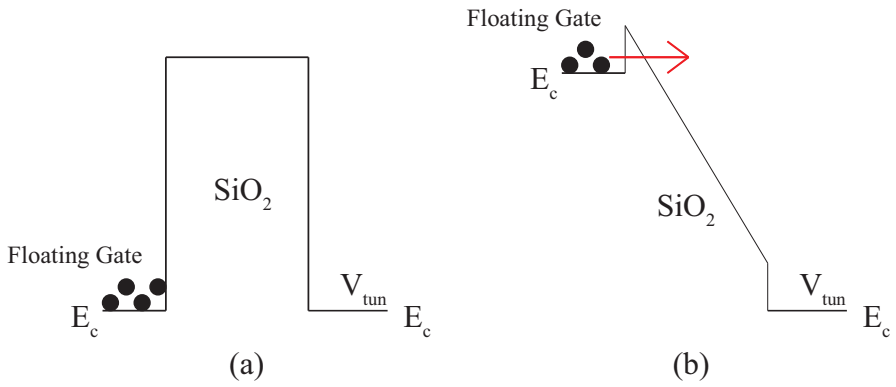


**Figure 7. Electron tunneling from [15]: (a): band diagram before applying a voltage across the barrier. The electrons cannot cross. (b): the voltage results in a decrease of the barrier thickness which allows electrons to go through.**

### 3.1.2 Injection

Hot-electron injection is the process that permits the addition of electrons to the floating gate. To do so, a large source-drain voltage is applied to the transistor while the current is flowing through the channel. The holes are accelerated towards the drain by the electrical field created through the channel. Some of the holes will collide with the ions located in the

drain-channel depletion region, creating electron-hole pairs. The electrons generated this way are then accelerated back toward the source. Most of these electrons are going to reach the well of the transistor. Some of them, qualified as "hot" electrons, gain enough energy to escape through the oxide and end up on the floating gate. These electrons negatively increase the charge on the floating gate, effectively decreasing $V_{offset}$. Figure 8 (from [16]) is an illustration of this process.

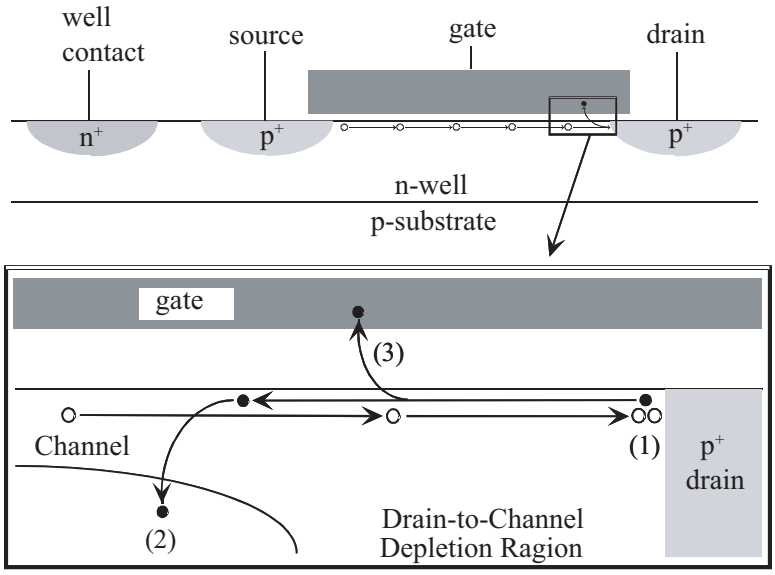


**Figure 8. Hot Electron Injection from [16]. Some "hot" electrons have enough energy to go through the oxide and end on the floating-gate.**

## 3.2 FPAA

### 3.2.1 Topology

The FPAA is divided into two types of circuitry fulfilling different functions. The first is the switch matrix, composed of interconnection elements. The second is composed of computational elements.

#### *3.2.1.1 Switch Matrix*

In the RASP chips, the switch matrix is built as an extended net composed of floating gate elements (FGEs) which can be programmed to connect computational elements together and achieve the desired circuit. Each floating gate transistor in this net has its source and

drain connected to a specific row and column, as shown on Figure 9 (from [17]). The row and column define the address of this specific FGE. With the two processes described earlier, it is possible to program precisely a specific FGE in the matrix. To do so, it is necessary to know the address as well as the target current to which the FGE should be programmed. The programming process will be described in greater detail in the next section. When a floating gate switch is fully on, the other elements sharing the same row or column are linked together. With this process, it is then possible to achieve numerous circuits that connect together the computational elements available on the chip.

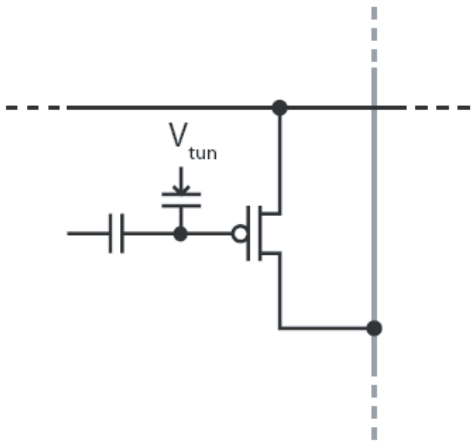


**Figure 9. Switch Element in the Switch Matrix from [17]. A floating-gate transistor is connected between a row and a column of the switch matrix. When programmed to be fully on, elements on the same row and columns are connected.**

#### *3.2.1.2 Computational Analog Block*

The second type of circuitry is the Computational Analog Block (CAB), which contains the computational elements. Depending on the FPAA, the CABs can be composed of various analog components, such as capacitors, operational transconductance amplifiers (OTAs), transistors, synapses etc. The inputs and outputs of each element are connected to rows of the switch matrix. As a consequence, it is possible to program the floating gate elements in the switch matrix in order to connect together several components in the same CAB or to elements in other CABs.

### *3.2.1.3 Mapping of the chip*

Generally, an FPAA is composed of several switch matrices and CABs, organized in rows and columns, allowing for bigger circuits. In Figure 10, the organization of a typical FPAA is shown. In this figure, four blocks of the RASP2.8 chip are visible. Each of them is composed of a switch matrix and a CAB. Only the vertical lines of the switch matrix are visible on this figure. The elements in the CABs of the RASP2.8a are classical analog components, such as pFets, nFets, OTAs and capacitors. As can be seen on this picture, several blocks can be connected together using vertical or horizontal global lines, which span more than one block. Thanks to this organization, it is possible to connect together computational elements in different blocks and create bigger circuits.

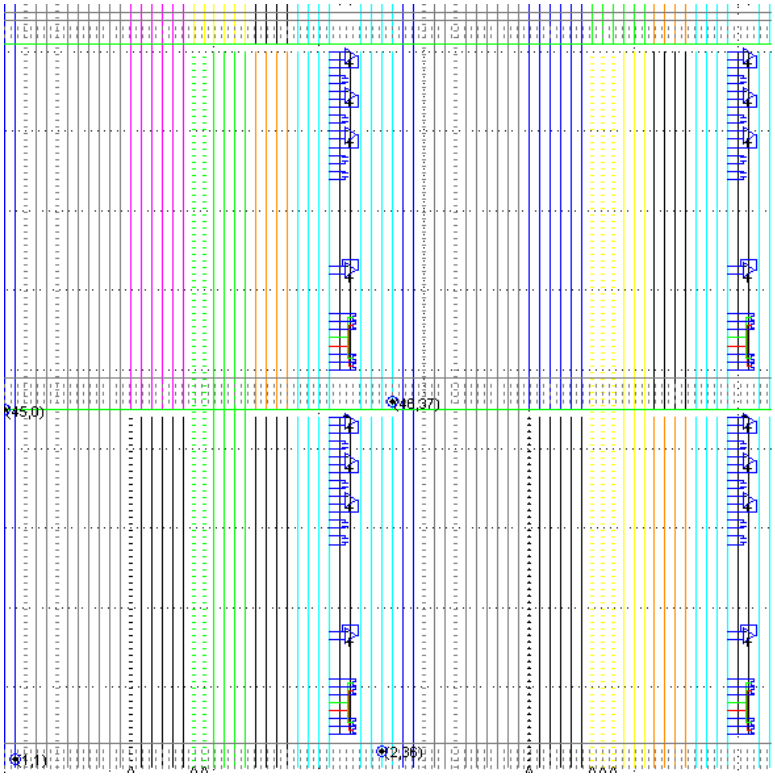


**Figure 10. FPAA architecture. On this figure, four blocks (composed of a switch matrix connected to a CAB) are represented. They are connected by horizontal and vertical global lines to allow the programming of bigger circuits.**

### 3.2.2 Array Programming

The switch matrix is composed of switches organized in rows and columns, as shown in Figure 11 (from [14]). With this architecture, it is possible to access a switch by its row and column address. This provides a fast way to program a selected switch [14].

#### *3.2.2.1 Rapid Programming*

For the tunneling process, the voltage across the MOS capacitor is set to a value suited for tunneling. Since several floating-gate transistors are connected to the same voltage control line, the tunneling is used to globally erase the charge on all the floating-gate elements. On the contrary, both the drain and the gate voltage have to be set to appropriate values in order to allow injection. As a consequence, on the intersection of the selected row and column, only one transistor will be in the right conditions to be programmed by hot-electron injection.

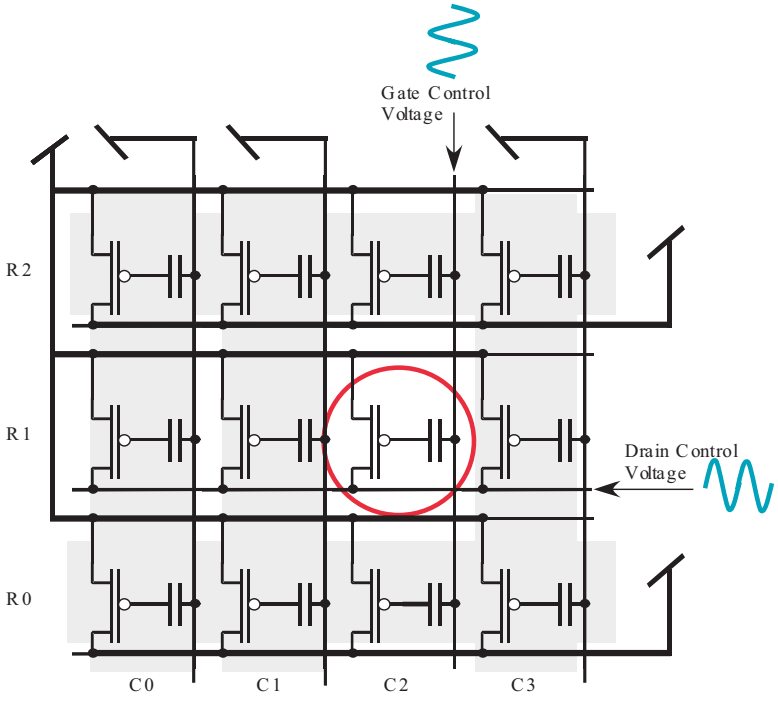


**Figure 11. FPAA programming Scheme from [14]. By applying the correct voltage on the specified row and column, it is possible to select and program one specific FGE on the chip.**

#### 3.2.2.2 *Indirect Programming*

Another design choice in the RASP device is the indirect programming [18]. In order to reduce the circuitry density, and thus the parasitic effects, an auxiliary transistor is used in the programming process. If only one transistor was used during programming, it would be necessary to disconnect it from the switch matrix and connect it to the programming structure to have control over its drain and gate voltages. In the indirect programming case, the transistor used in the circuitry (on the left on Fig. 12) (Figure from [19]) does not need to be disconnected. It shares its floating-gate with the transistor connected to the programming circuitry (on the right). When charges are added or removed by tunneling and injection on the programmed transistor, they lie on the common node and the second transistor's floating-gate is programmed too.

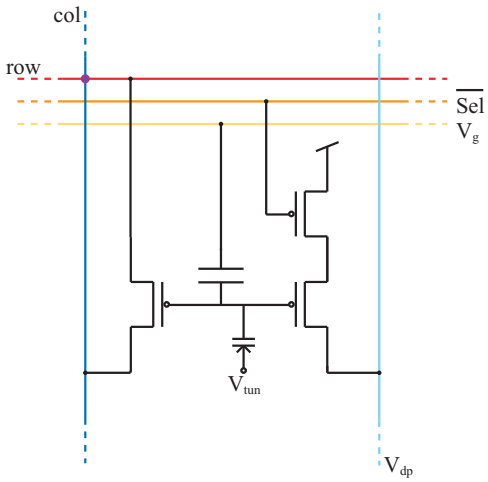


**Figure 12. Indirect Programming from [19]. The transistors on the right are connected to the programming circuitry, whereas the transistor on the left is the one being used in the circuit.**

## 3.3 Tools

In order to make the FPAA more user-friendly and accessible to people that have no knowledge of the FPAA architecture, and even little knowledge of circuit design, a series of tools have been developed by the lab that allow one to operate the FPAA at a higher level of conception. With these tools, it is possible to start the design of the desired circuit using

Simulink. This software, familiar to engineers and students, contains libraries of functional blocks that can be connected together to create circuits. Thanks to the tools developed in the lab, the process leading to the actual row and column addresses of the switches on the FPAA is hidden to the user, and no additional knowledge is required to implement and test the circuit designed in Simulink. In a first step, the Simulink design of the circuit is converted into a classical Spice file with the *Sim2spice* tool. The resulting file describes the different analog components and their connections and can be simulated in the classical Spice environment. In a second step, the GRAPSER tool creates a netlist containing all the different switches and their targeted values to be programmed on the specified FPAA. This netlist can be directly used in Matlab and serves to program the chip, thanks to the programming code and the micro-controller. Finally, the same netlist can be input in a visual interface, the RAT, in order to visualize the circuit generated on the FPAA. This last tool is useful to check that a circuit is correct, or simply to know which CABs and CAB elements are used and what the input/output connections are. A testing board developed by the lab allows the testing of the circuit.

### 3.3.1 Sim2spice

This software, developed by Csaba Petre and Craig Schlottmann, is coded in Matlab and generates a Spice file from a circuit designed in Simulink [20]. The Simulink tool is part of the Mathworks Matlab toolbox. It allows the user to draw their circuit using different blocks saved in libraries that can be customized. The actual analog implementation of each block is hidden to the user, who only has to connect the necessary blocks together to obtain the desired function. Students in the lab developed several blocks that are often used in the FPAA applications, such as VMMs, V-to-I and I-to-V blocks. Using this high-level description of the circuit and files describing the analog implementation of the blocks, *Sim2spice* generates an output file written in the classical Spice syntax. The output file includes a description of the subcircuits and of the input/output connections, which are added automatically in a way that does not interfere with Spice and that can later be understood by the

GRASPER software.

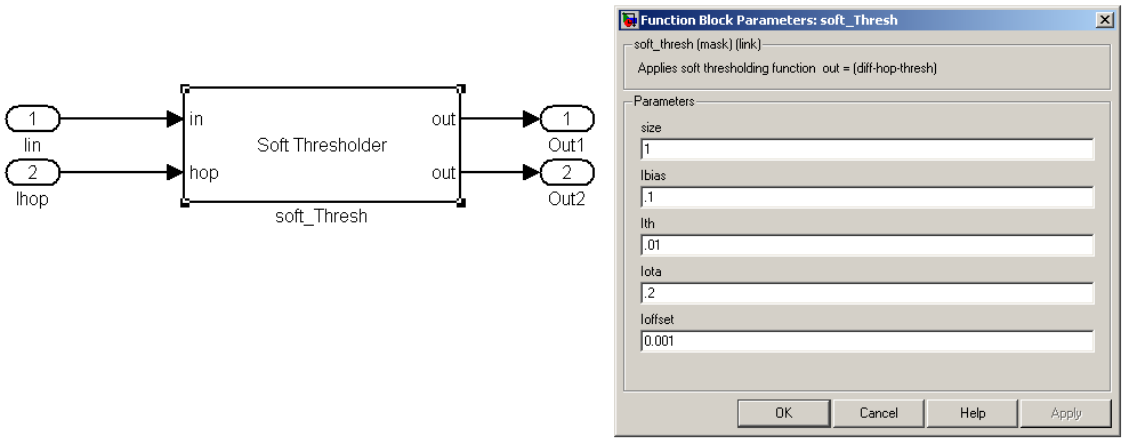


**Figure 13. Simulink Block for the Soft-Threshold. This block was created to implement the soft-threshold operator used in this research. Associated files permit *Sim2Spice* to build the corresponding circuit netlist for Spice. The user can define the number of inputs and outputs needed (size), the reference current in output (Ibias), the desired threshold current (Ith), the bias current necessary for the OTA (Iota) and the offset to adjust for eventual mismatch (Ioffset).**

For the purpose of this research, Sam Shapero and I developed a soft-threshold block and added it to the custom library, LCA, in Simulink. Figure 13 shows the block as it appears in Simulink and the parameters that the user can define for his own design.

### 3.3.2 GRASPER

This software was developed by Faik Baskaya for his PhD [21, 22, 23]. This tool is used to rout a circuit described in a Spice file onto a specified FPAA. The components in the circuit are placed in the most efficient way and routed to output pins. The output file is a netlist containing the different switches, characterized by their row and column address and the value they should be programmed to. The resulting file can be directly used in Matlab to program the FPAA.

This software can support any type of FPAA architecture, described in an auxiliary device file. This auxiliary file contains information such as the number of rows and columns per CAB, the CABs' elements and their location, etc. Moreover, the user has the option to specify which input/output connections to use, as well as which CABs or which elements

in a CAB to use in priority. If none of these are precised, the GRASPER will pick them automatically and return an optimal implementation of the circuit.

### 3.3.3 RAT

The RAT, or Routing and Analysis Tool, was developed in Matlab by Scott Koziol and David Abramson. It provides a means to visualize any circuit described by a switch netlist, such as the one generated by GRAPSER. It is useful to visualize the general aspect of a circuit and its repartition on the different CABs of the FPAA. For instance, it permits one to determine which elements are being used, or which I/O pins were selected by GRASPER if not specified by the user. It is also extremely valuable when debugging a circuit because it prevents one from having to do the analysis by hand. It is possible to zoom in and out and navigate through the circuit, as well as to add or delete nodes interactively. Moreover, like for the GRASPER tool, this interface can be generalized to different FPAA architectures. I have helped to adapt the code in order to use the RAT on the new RASP 2.9a.

Figure 14 shows an example of the RAT interface for the routing of a simple OTA. In this example, the programmed switches are circled in blue. The red lines show the connections resulting from the activation of these switches. In this example, one input of the OTA is connected to the global line *Vdd* (vertical blue line on the left), the other input is connected to an I/O pin (horizontal dashed line lt<0> at the bottom) and its output to another I/O pin (horizontal dashed line rt<0> at the bottom).

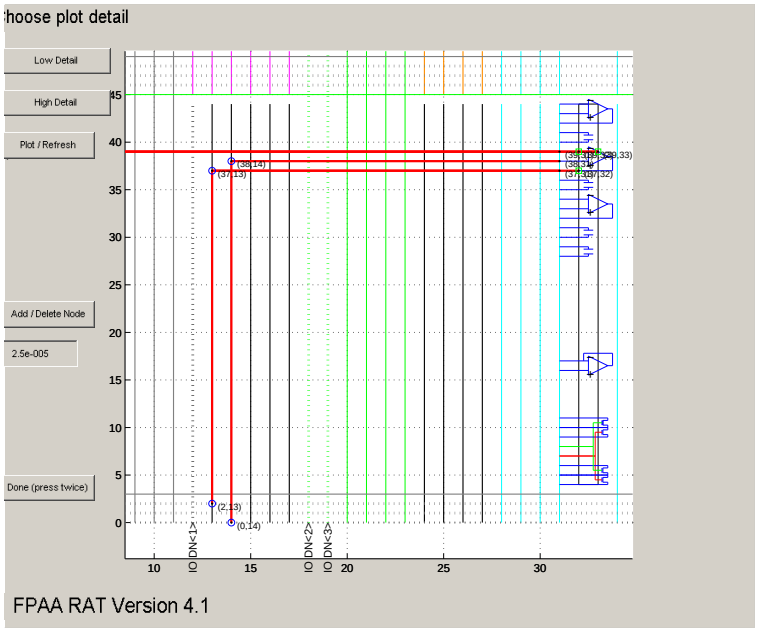


**Figure 14. Example of a circuit visualized using the RAT. The red lines show the connection created by the switches that are present in the list to program. The OTA inputs and output are connected to Vdd and to I/O pins. The RAT allows to make sure that a list correspond to the desired circuit and helps for debugging.**

### 3.3.4 The Testing Board

In order to program, get measurements, and control the chip, a printed circuit board was developed by the CADSP lab. It is connected to a computer and powered via a USB port. A micro-controller on the board allows the user to use Matlab commands to control the board. The board contains DAC and ADC pins, as well as audio input/output amplifiers and jacks.

From the user point of view, every I/O pin present in the circuit can be accessed on the board via pins. To program the desired circuit on the RASP, the user sends commands directly from Matlab to the board. Using wire jumpers, they can then connect I/O pins together, or connect them to DACs and ADCs in order to set voltages or read measurements. The voltage setting and current reading or any other operation can be written in Matlab and sent to the chip. Finally, the results are sent back to the computer by the board or the

measuring device, and can be stored and treated in Matlab, in the form of vectors. They can then be plotted or used for other computational operations.

Figure 15 is a picture of the board with wires to connect different I/O pins.

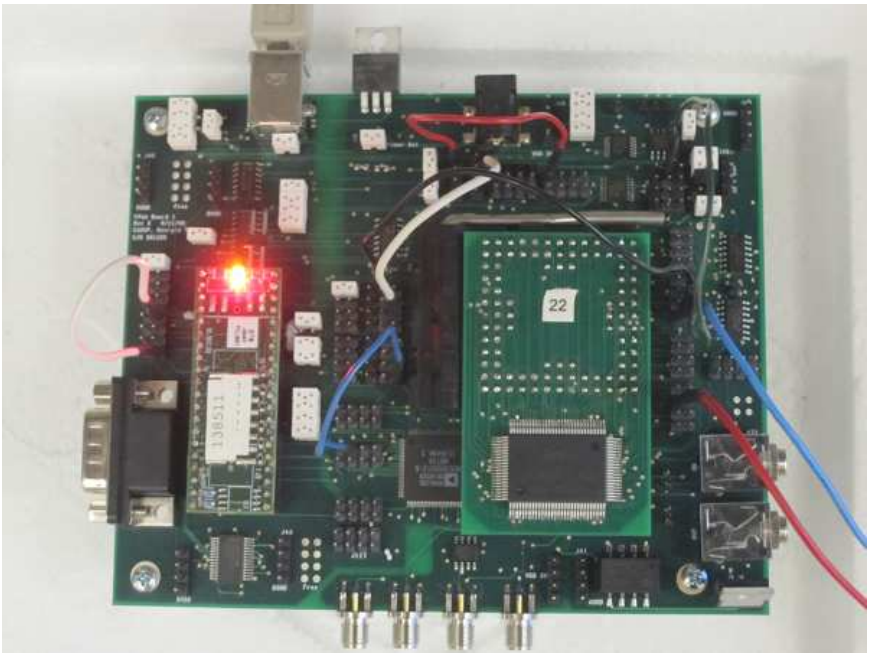


**Figure 15. Picture of the Testing Board. The RASP chip is on the right. A microprocessor, on the left, transfers commands sent from the computer to the board. The pins on the board permit to connect I/O to measurement devices.**

# CHAPTER 4

# SYSTEM ARCHITECTURE

This chapter describes the analog implementation of the LCA. First, the global architecture with its different blocks will be presented. Second, subsections will go into the details of each block, the functions they are expected to realize and how they integrate to the global system. Finally, an example will illustrate what function each block is supposed to fulfill.

## 4.1 Global System

### 4.1.1 Choices made

The signals of interest in the LCA, as well as the inputs and outputs, are continuous by nature. In analog circuits, continuous signals are generally represented by either a voltage or a current. As will be described later, the LCA circuit uses Vector Matrix Multipliers (VMMs) that are designed to operate on sub-threshold currents. As a consequence, a natural choice for the system was to use currents to carry the information. To be more precise, when considering a signal $s(t) = \begin{bmatrix} s_1(t) & \cdots & s_N(t) \end{bmatrix}^T$, $N$ currents will be used to represent each of the components $s_n(t)$ in the vector.

Another choice made when designing the proposed architecture concerns the soft-threshold. This block needs to introduce the non-linearity of the algorithm. However, in order to consider positive as well as negative inputs, it is necessary to have two non-linearities, one for positive inputs and one for negative inputs. To do so, two solutions are possible. The first consists of creating an entirely differential block, which implements the two non-linearities. The alternative solution is to implement only the positive half of the soft-threshold function. This second solution was chosen for this research, due to limited space and for simplicity. In this case, in order to keep the possibility of having both

negative and positive inputs, nodes and outputs, it becomes necessary to double the number of nodes in the system. For each node $\Phi_m = \begin{bmatrix} \phi_{m1,}(t) & \cdots & \phi_{m,N}(t) \end{bmatrix}^t$, its symmetric $-\Phi_m = \begin{bmatrix} -\phi_{m,1}(t) & \cdots & -\phi_{m,N}(t) \end{bmatrix}^t$ is added to the dictionary. The matrix $\Phi$ now contains $2M$ nodes, or columns. By doing so, the system will contain only positive outputs, and only one output in each pair ($\Phi_m$ and $-\Phi_m$) will be non-zero. The non-zero outputs will be equal to the absolute value of the output coefficient corresponding to the signed node. By doing so, we limited the soft-threshold and the outputs to be only positive, but the sign information is kept by looking at which node is active: the original node or its symmetric.

For simplicity, rename the original set of nodes, with index from 1 to $M$: $\tilde{\Phi}_1, \tilde{\Phi}_2, \ldots, \tilde{\Phi}_M$. And name the new set of $2M$ nodes $\Phi_1, \Phi_2, \ldots, \Phi_{2M}$, with:

for any $m = 1, \ldots, M, \Phi_{2m-1} = \tilde{\Phi}_m$ (renaming the original nodes)

for any $m = 1, \ldots, M, \Phi_{2m} = -\Phi_{2m-1} = -\tilde{\Phi}_m$ (adding the symmetric nodes to the new set)

It is important for the following sections to recall that the nodes of odd number correspond to the original dictionary elements and the nodes of even number to their symmetric.

### 4.1.2 Architecture

The characteristic ODE of the LCA in (4) can be rewritten in a way more suited for circuits:

$$\begin{aligned} \tau \dot{u}(t) + u(t) &= b - (\Phi^t \Phi - I)\, a(t) \\ a(t) &= T_\lambda(u(t)) \end{aligned} \tag{6}$$

This ODE is composed of several simple operations which can be executed by independent functional blocks. The following sections present these different blocks and their respective functions.

Figure 16 shows a block diagram of the LCA circuit proposed. The first operation to perform is the vector matrix multiplication $b = \Phi^t s$. This operation is realized by the first VMM on the figure. Another block is needed to create the inhibition coefficients between the nodes, $h(t) = (\Phi^t \Phi - I)a(t)$. This operation is also achieved by a VMM (second block from the left on the figure). The last functional block realizes the threshold function, which

is essential in any neural network and corresponds to the operation $a(t) = T_\lambda(u(t))$. The low-pass filter operation on the internal variables ( $\tau\dot{u}(t) + u(t)$ ) is implicitly implemented by the analog components present in the circuit. Indeed, the VMMs are composed of several capacitive devices. Combined with the resistivity of the wires, the VMMs induce a *RC* time constant in the system, creating the desired low-pass filter behavior as the number of nodes increases. The circles on the figure show which variables are differential: that is, the variables which can be either positive or negative. Finally, the different blocks are linked together in a way that leads to the desired ODE.

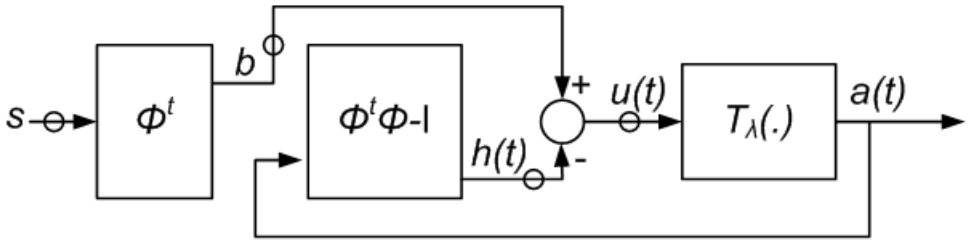


**Figure 16. LCA Circuit Architecture. The circuit proposed to implement the LCA is composed of three blocks: two VMMs and a soft-thresholder. The low-pass filter operation is implicitly performed by the VMMs. The circle on the figure represent the differential currents on the current circuit architecture.**

From Figure 16, the low-pass filter operation and the current subtraction lead, in the Laplace domain, to:

$$u(s) = \frac{b - h(s)}{1 + s\tau}$$

where $\tau$ is the RC time constant of the VMM.

After expanding, we get:

$$u(s) = \frac{b - (\Phi^t\Phi - I)a(s)}{1 + s\tau}$$
$$\tau s u(s) + u(s) = b - (\Phi^t\Phi - I)a(s)$$

After transforming back to the time domain, we obtain:

$$\tau\dot{u}(t) + u(t) = b - (\Phi^t\Phi - I)a(t)$$

which corresponds exactly to the expected ODE in (6).

## 4.2 Individual Blocks

As previously stated, the system was implemented using current mode devices, which allowed for minimum size implementation. As a consequence, the inputs and the measured outputs of the LCA circuit are currents. However, the FPAA chip sets and measures voltages more easily than currents. To deal with this, voltage-to-current and current-to-voltage converters were used in the input and output stages of the system, and the outputs were measured through an off-chip picoammeter.

### 4.2.1 Inputs

From Matlab, commands can be sent to the chip to set voltages in the inputs of the system. A voltage-to-current block was used to transform these voltages into currents. Ideally, the voltage to current relationship of this block is linear. In reality, by a careful characterization of this block, it is possible to compensate for eventual non-linearities.

### 4.2.2 Outputs

Since the system's variables are currents and it is easier to measure voltages on the chip, the output blocks must realize the inverse operation. This was achieved by a current-to-voltage converter. Once again, the input-output relationship of this block is expected to be linear.

### 4.2.3 Driving VMM

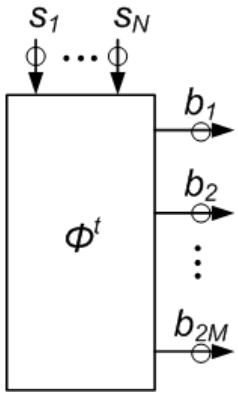


**Figure 17. Block representation of the Driving VMM. The DVMM computes the driving inputs, which represent how close the input signal of dimension $N$ is to each of the $M$ nodes.**

The first VMM that intervenes in the system is named the Driving VMM. It must achieve the computation of the driving inputs. An input is represented by $N$ currents: $I = \begin{bmatrix} I_1 & \cdots & I_N \end{bmatrix}^t$. The output of this block is the vector of driving inputs $b = \begin{bmatrix} b_1 & \cdots & b_{2M} \end{bmatrix}^t$. The driving inputs account for how well the input matches each node. The closer the input is to one node, the higher the driving value for this node will be. This can be expressed in terms of an inner product of the form $b_m = \langle \Phi_m, I \rangle = \Phi_m^t I$, where $\Phi_m$ is the vector $\Phi_m = \begin{bmatrix} \phi_{m,1} & \cdots & \phi_{m,N} \end{bmatrix}^t$ corresponding to one of the $2M$ nodes.
This operation can be written in a matrix form:

$$\begin{bmatrix} b_1 \\ \\ \vdots \\ \\ b_{2M} \end{bmatrix} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,N} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,N} \\ \phi_{3,1} & \phi_{3,2} & \cdots & \phi_{3,N} \\ \vdots & \vdots & \cdots & \vdots \\ \phi_{2M,1} & \phi_{2M,2} & \cdots & \phi_{2M,N} \end{bmatrix} \begin{bmatrix} I_1 \\ \\ \vdots \\ \\ I_N \end{bmatrix}$$

It is obvious that a VMM of size $2M \times N$ can be used to realize this operation (see section 5.2).

Since the inputs and outputs of this block can be either positive or negative, but the currents in the VMM can only be positive, it is necessary to use a differential structure in input and output of this block. The method to implement the differential structure will be presented in section 5.2.3. The idea is to separate the currents into a positive and a negative half, and recenter them around a reference value $I_{ref}$. Both resulting currents are positive and can be used in the VMM.

### 4.2.4 Threshold function

This block was designed to implement only the positive half of the soft-threshold function described in section 2.3.2, plotted again on Figure 19. The inputs of the soft-threshold block are the driving inputs $b$ obtained with the Driving VMM and the Hopfield inputs (or feedbacks) $h(t)$, coming from the Hopfield VMM. Since these currents are differential,

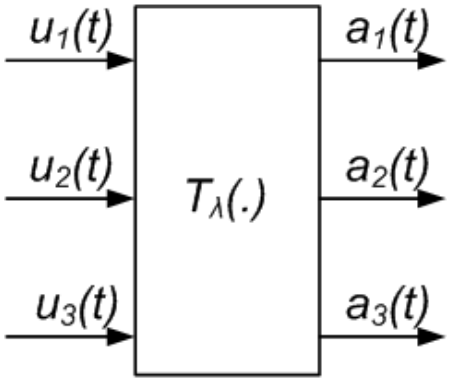


**Figure 18. Block representation of the Soft-Thresholder. Only the positive half has to be implemented provided that we double the number of nodes in the dictionary. The outputs are zero below the threshold, and equal to the input minus the threshold above the threshold.**

they are composed of a positive and a negative part. The soft-threshold block recovers the effective current by subtracting the two parts $Iin_+ - Iin_-$. As can be seen in Figure 19, the outputs of this block are always positive. To get the thresholded outputs, the exact function that the soft-threshold block has to implement is:

$$I_{out} = \max(Iin_+ - Iin_- - I_{th} + I_{ref}, 0) \tag{7}$$

The output needs to be repeated to allow the measurement of the $2M$ global outputs of the system $a_m$, while also looping the output currents back into the inputs of the Hopfield VMM.

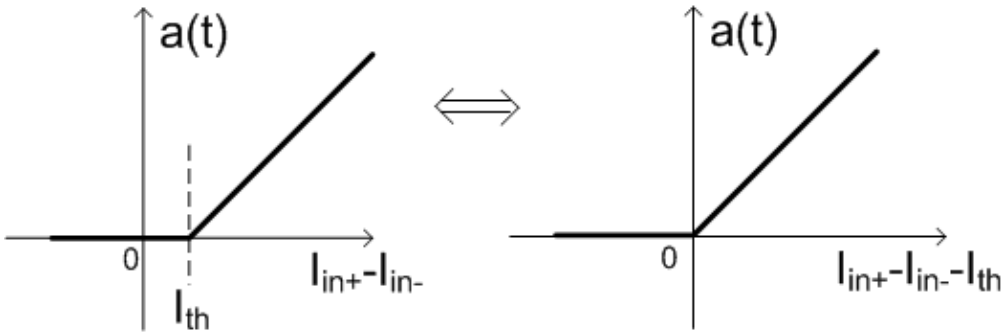


**Figure 19. Plots of the soft-threshold function. The two plots are equivalent. Only the positive half of the soft-threshold is plotted.**

To perform the operation $b - h(t)$, the positive half of the differential current coming from the Driving VMM is connected to the positive input of the soft-threshold, while the negative half is connected to the negative input. On the contrary, the positive half of the differential current coming from the Hopfield VMM is connected to the negative input of the soft-threshold, and the negative half to the positive input. This yields:

$$Iin_+ = b_+ + h_-$$
$$Iin_- = b_- + h_+$$

If initially the number of nodes was doubled (by adding its symmetric to each original node in the dictionary) this block will also ensure that only one node in each couple $(\Phi_{2m-1}, \Phi_{2m})$ will be non-zero. Indeed, if a particular node $\tilde{\Phi}_m$ leads to the output $\tilde{a}_m$ in the LCA algorithm, then its symmetric would lead to the output $-\tilde{a}_m$. One of these two values is negative and it will be set to zero by the soft-threshold block. Consequently, the node $\tilde{\Phi}_m$ appears only once in the global output and with the correct sign.

The output of this block is also the global solution to the optimization problem (3). This solution can be simply written as a sum of the output coefficients times the corresponding nodes:

$$a(t) = \sum_{m=1}^{2M} a_m \Phi_m$$

$$a(t) = \sum_{m=1}^{M} \left( a_{2m-1} \tilde{\Phi}_m - a_{2m} \tilde{\Phi}_m \right) \quad \text{with} \quad a_{2m-1} \text{ or } a_{2m} = 0$$

### 4.2.5 Hopfield VMM

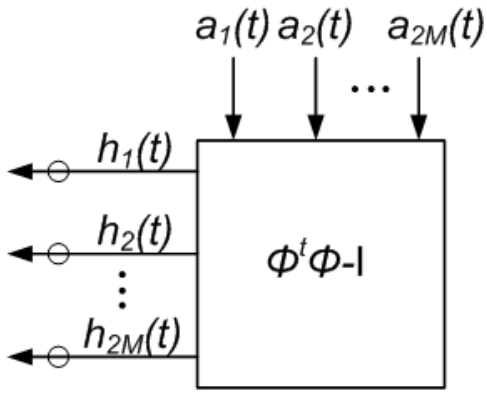


**Figure 20. Block representation of the Hopfield VMM. The HVMM computes the feedbacks between the nodes.**

The mathematical function that this block realizes is the multiplication by the matrix $\Phi^t\Phi - I$ of the current outputs $a(t)$. The outputs are the feedbacks that are fed back into the inputs of the soft-threshold: $h = (\Phi^t\Phi - I)a$:

$$\begin{bmatrix} h_1 \\ \vdots \\ h_{2M} \end{bmatrix} = \begin{bmatrix} 0 & \langle\Phi_1,\Phi_2\rangle & \langle\Phi_1,\Phi_3\rangle & \cdots & \langle\Phi_1,\Phi_{2M}\rangle \\ \langle\Phi_2,\Phi_1\rangle & 0 & \langle\Phi_2,\Phi_3\rangle & \cdots & \langle\Phi_2,\Phi_N\rangle \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \langle\Phi_{2M},\Phi_1\rangle & \langle\Phi_{2M},\Phi_2\rangle & \cdots & 0 & \langle\Phi_{2M},\Phi_{2M}\rangle \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{2M} \end{bmatrix}$$

This operation can be achieved by a VMM of size $2M \times 2M$. This VMM performs exactly the typical operation achieved in a Hopfield network: that is, the computation of the feedbacks in between the nodes. In a typical Hopfield network, there is no feedback from one node to itself (the coefficients on the diagonal are equal to zero), and the feedback from node i to node j is equal to the feedback from node j to node i (from the symmetry of the matrix: $G_{ij} = \left\langle \Phi_i, \Phi_j \right\rangle = G_{ji}$, for $i \neq j$ ). For this reason, this block will be called Hopfield VMM from now on.

For this particular VMM, the input is a vector of size $2M$ whose components are all

positive, since they are the outputs $a_1$ to $a_{2M}$ of the soft-threshold block previously described. However, the outputs of this block can be either positive or negative. For this reason, this VMM has to be built with single inputs and differential outputs. The analog implementation of such a VMM will be presented in section 5.2.3.

## 4.3 Example

This section presents an example of the operations performed by each block in order to get a better sense of how the system works.

Consider a two-dimensional input $I_{in} = \begin{bmatrix} I_1 & I_2 \end{bmatrix}^t$ and three nodes normed in $\mathbb{R}^2$ $\tilde{\Phi}_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^t$, $\tilde{\Phi}_2 = \begin{bmatrix} 0.6 & 0.8 \end{bmatrix}^t$ and $\tilde{\Phi}_3 = \begin{bmatrix} 0 & 1 \end{bmatrix}^t$.

The first step consists of constructing the six nodes
$\Phi_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^t, \Phi_2 = \begin{bmatrix} -1 & 0 \end{bmatrix}^t, \Phi_2 = \begin{bmatrix} 0.6 & 0.8 \end{bmatrix}^t,$
$\Phi_3 = \begin{bmatrix} -0.6 & -0.8 \end{bmatrix}^t, \Phi_5 = \begin{bmatrix} 0 & 1 \end{bmatrix}^t, \Phi_6 = \begin{bmatrix} 0 & -1 \end{bmatrix}^t$
and to store them in a matrix of size $2 \times 6$:

$$\Phi = \begin{bmatrix} \Phi_1 & \Phi_2 & \Phi_3 & \Phi_4 & \Phi_5 & \Phi_6 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0.6 & -0.6 & 0 & 0 \\ 0 & 0 & 0.8 & -0.8 & 1 & -1 \end{bmatrix}$$

This matrix corresponds to the Driving VMM described earlier. When multiplying the inputs with this matrix, the resulting outputs are the driving inputs: $b = \Phi^t I_{in}$. In this case, we get:

$b_1 = \Phi_1^t I = I_1$

$b_2 = -I_1$

$b_3 = 0.6I_1 + 0.8I_2$

$b_4 = -0.6I_1 - 0.8I_2$

$b_5 = I_2$

$b_6 = -I_2$

The outputs of this block, $b$, as well as the outputs of the Hopfield VMM, $h$, are sent to the input of the soft-threshold block, to form the thresholded outputs $a_m$, $m = 1, \ldots, 6$, which are only positive or null.

They are then sent to the Hopfield VMM of size $6 \times 6$ to compute the 6 differential Hopfield inputs $h_1, \ldots, h_6$:

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0.6 & -0.6 & 0 & 0 \\ -1 & 0 & -0.6 & 0.6 & 0 & 0 \\ 0.6 & -0.6 & 0 & -1 & 0.8 & -0.8 \\ -0.6 & 0.6 & -1 & 0 & -0.8 & 0.8 \\ 0 & 0 & 0.8 & -0.8 & 0 & -1 \\ 0 & 0 & -0.8 & 0.8 & -1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix}$$

Finally, the outputs of the entire system are also the $a_m$, for $m = 1, \ldots, 6$, which are always positive, such that:

$$s = a_1\Phi_1 + a_2\Phi_2 + a_3\Phi_3 + a_4\Phi_4 + a_5\Phi_5 + a_6\Phi_6$$

$$s = a_1\tilde{\Phi}_1 - a_2\tilde{\Phi}_1 + a_3\tilde{\Phi}_2 - a_4\tilde{\Phi}_2 + a_5\tilde{\Phi}_3 - a_6\tilde{\Phi}_3$$

with $(a_1 = 0 \; or \; a_2 = 0)$ and $(a_1 = 0 \; or \; a_2 = 0)$ and $(a_5 = 0 \; or \; a_6 = 0)$

All the information needed is recovered, that is to say, which nodes are active, with which weight and which sign.

In section 5, this example will be used, keeping only the original nodes and limiting the inputs to be in the first quadrant for a proof of concept (see Fig.21).

The norm of the nodes is chosen to be one. The input currents will be between 0 and $20nA$ in this example. The threshold current $I_{th}$ is chosen to be $10nA$ and the reference current is $I_{bias} = 100nA$.

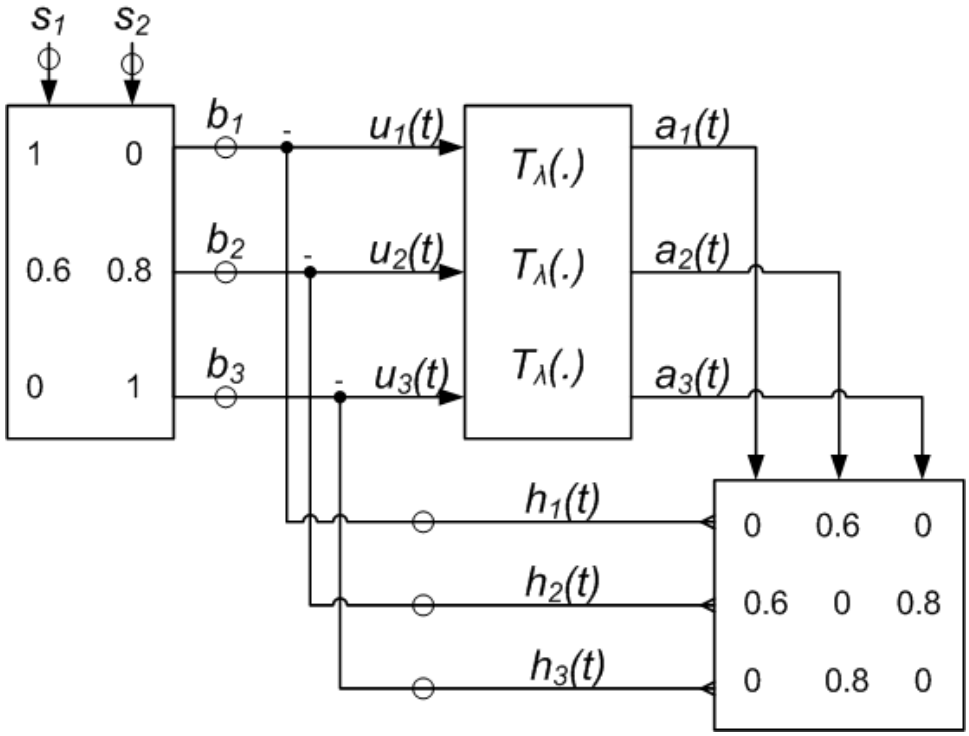


**Figure 21. Example of a 2x3 LCA system. The nodes are chosen to be in the first quadrant as well as the inputs for a proof of concept.**

# CHAPTER 5

# RESULTS

In the previous chapter, the architecture of the system was presented and each part was described in a functional way. This chapter will concentrate on the analog implementation of each part and the results obtained on the FPAA.

In a first section, the methodology followed for the testing of the circuit will be described. In a second section, a general implementation of Vector Matrix Multipliers using floating-gates transistor will be presented. In the last part, the analog design of each block will be presented and experimental results will be given.

## 5.1 Methodology

The experimental approach for testing the circuit began with characterizing each block of the LCA circuit independently in order to determine their dynamic ranges. This is a necessary step before a full system can be built.

In a first step, the LCA circuit to test was drawn in Simulink, using functional blocks of a custom library, named LCA, which was created for the purpose of this research. This library contains the soft-threshold block, different VMMs and the V-to-I and I-to-V converters. Figure 22 shows the Simulink block representation of the LCA circuit. The size and parameters of these blocks can be defined by the user.

Then, the program *Sim2spice* was used on this Simulink model to get a description of the circuit in Spice for the RASP2.8a chip. Once the Spice file obtained, it was used in input of the GRASPER software to get the switch netlist to be programmed on the FPAA. For a LCA of size $2 \times 3$ (2 dimensional inputs and three output nodes), it was necessary to manually modify the I/O connections selected by *Sim2spice* in order to get a circuit that could be routed on the RASP2.8 chip. The board used for testing is the one presented in

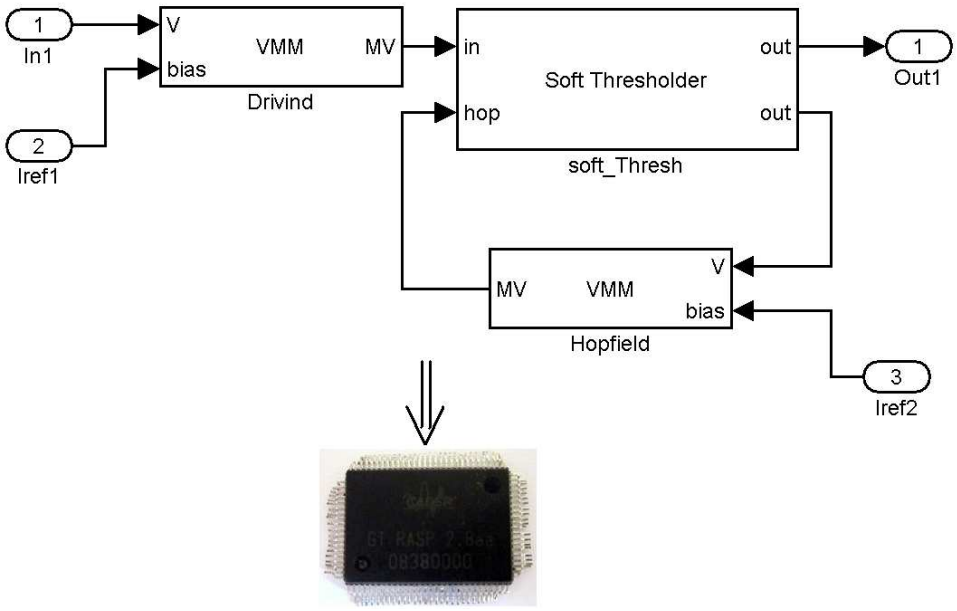


**Figure 22. Simulink block representation of the LCA. The circuit was drawn in Simulink using a custom library. The tool chain developed in the lab permits to readily obtained a list of switches and target values to be programmed on the target RASP chip.**

section 3.3.4. The list of routing elements given by GRASPER was used to program the circuit on the chip. All the testing routines were written in Matlab, and the results were exported and plotted back in Matlab.

## 5.2 VMM

Vector-Matrix Multipliers are very useful computational blocks. When operating in current mode, they perform the operation $b = \Phi a$, where both inputs and outputs are currents. The LCA circuit proposed contains two VMMs. The VMM implementation that this section describes uses floating-gate transistors to implement the weights in the matrix $\Phi$. This allows for a minimum size implementation [12].

### 5.2.1 Current Mode VMM

The generic current mode VMM presented performs the operation:

$$\begin{bmatrix} b_1 \\ \\ \vdots \\ b_M \end{bmatrix} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,N} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,N} \\ \vdots & \vdots & \cdots & \vdots \\ \phi_{M,1} & \phi_{M,2} & \cdots & \phi_{M,N} \end{bmatrix} \begin{bmatrix} a_1 \\ \\ \vdots \\ a_N \end{bmatrix}$$

where inputs and outputs are sub-threshold currents.

In order to create the different weights in the matrix Φ, floating-gate transistors are used. Figure 23 shows how to create a single weighting element: $b = \phi_{1,1}a$. The floating-gate transistor on the left and the OTA are used to convert the input current $a$ into a voltage, which is then transformed back into a current by the transistor on the right. The difference of charges present on the two floating-gates will lead to a weight $\phi_{1,1}$, according to (8). By changing the amount of charge on the gate of the second transistor, it is possible to get the desired weight.

$$\phi_{1,1} = e^{-\frac{C_1 - C_0}{U_T}} \tag{8}$$

The charge $C_0$ programmed on the input floating-gate and the bias of the OTA should be large enough to permit the maximum needed input current to pass.

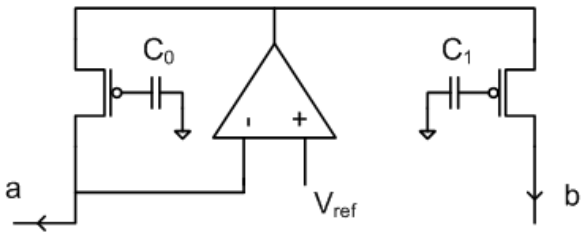


**Figure 23. Implementation of a single multiplier element. The amount of charge programmed on the two floating-gate transistors determines the ratio between the input and the output currents.**

After testing this element on the FPAA, the results shown on Figure 24 were obtained. The output current is plotted as a function of the input current, for several currents programmed on the second transistor. As expected, a wide range of weights can be obtained.

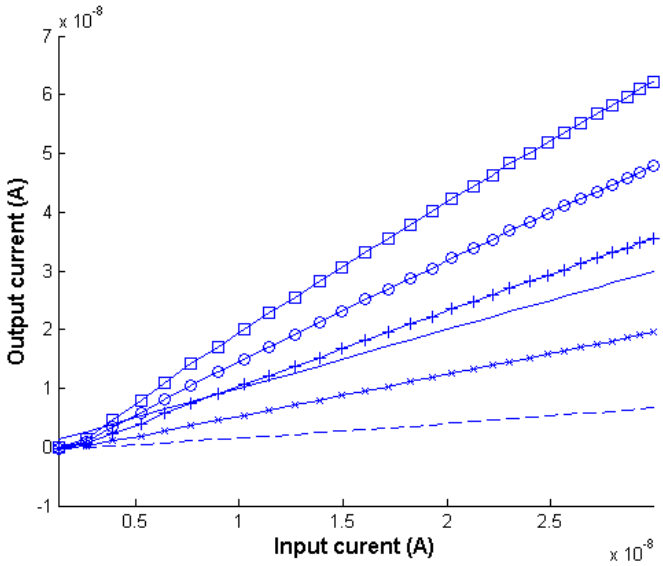


**Figure 24. Experimental result of a single multiplier element. The second floating-gate transistor was programmed to five different currents. The resulting output currents are plotted against the input current. The plain line represents a one-one relationship.**

In a second step, it is possible to create several weighted outputs from a single input: $\begin{bmatrix} b_1 & b_2 & \cdots & b_M \end{bmatrix}^t = \begin{bmatrix} \phi_{1,1} & \phi_{2,1} & \cdots & \phi_{M,1} \end{bmatrix}^t \begin{bmatrix} a \end{bmatrix}$. To do so, the circuit shown in Figure 25 is used. Here again, the relative amounts of charge on the output FGEs and the input FGE provide the different weights in the vector $\Phi_1$, according to the same relation as in (8).

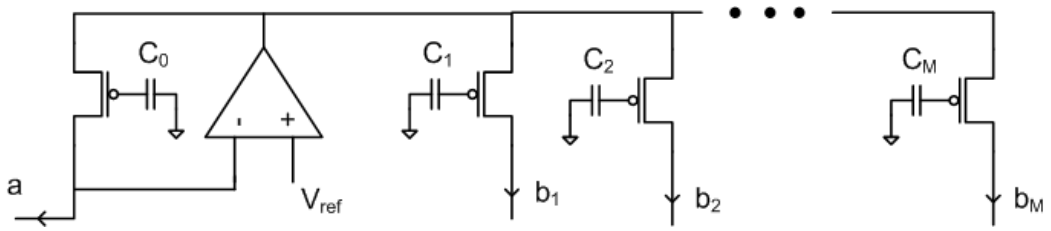


**Figure 25. Implementation of a multiplier with one input and *M* outputs. The amount of charge programmed on one of the *M* output floating-gate transistors determine the slope of the linear input/output relationship for this particular output.**

Finally, in order to get the entire matrix multiplication, the summation $b_i = \sum_{k=1}^{N} \phi_{i,k} a_k$ has to be obtained for each output. The summation of different currents is easily achieved by connecting the different output wires together. Figure 26 shows this process.

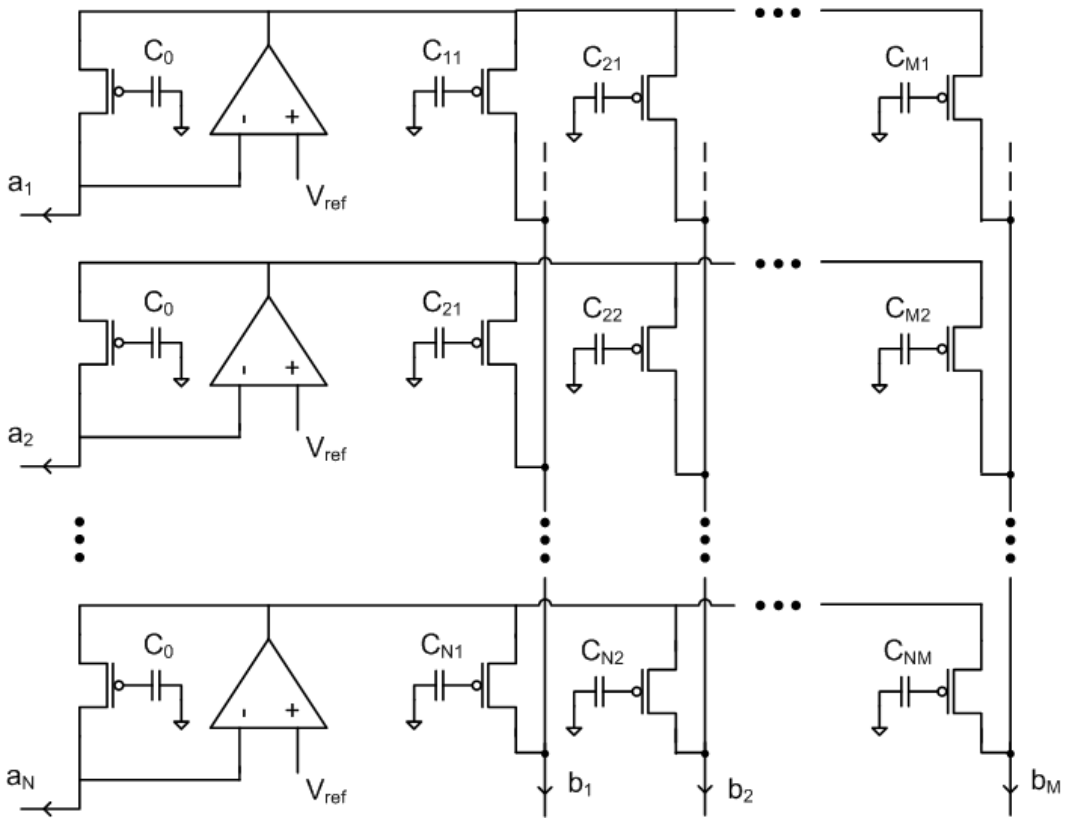


**Figure 26. Implementation of a multiplier with $N$ inputs and $M$ outputs. The amount of charge programmed between of the $NM$ floating-gate transistors in output and the input floating-gate on the same row determines one of the weights of the matrix. The currents are summed up along the columns to generate the output.**

### 5.2.2 Example

Consider a $2 \times 1$ VMM. This means that there are two input currents and one output current satisfying the following relationship: $I_{out} = w_{11}I_{in1} + w_{21}I_{In2}$

According to the previous section, this is achieved by the block on Figure 27.

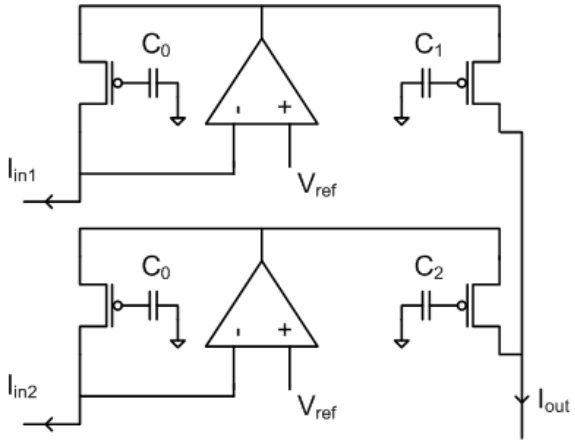


**Figure 27. Circuit Design of a 2x1 VMM. This circuit is composed of 2 inputs, 1 output and 2 weights.**

Figure 28 shows experimental results of this structure on the FPAA.

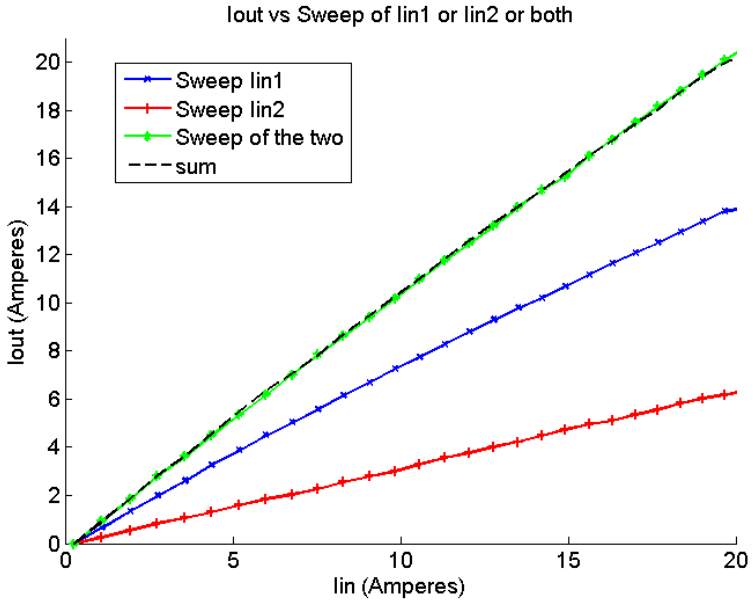


**Figure 28. Experimental result of a 2x1 VMM. First, the two inputs were first swept independently, resulting in the two bottom-most lines. The inputs were then swept simultaneously, yielding an output equal to the sum of the first two lines.**

The first input current was swept linearly while the second input was maintained constant. The slope of the resulting output (in blue with × symbols) corresponds to the first weight $w_{11}$. Similarly, a sweep of the second input while the first input was kept constant produces a line with slope $w_{21}$ (in red with + symbols). On the third curve (in green with * symbols), the two inputs were swept simultaneously with $I_{in1} = I_{in2}$. Finally, the dashed curve shows the summation of the first two lines point-by-point. As expected the slope is identical for the last two curves: $I_{out} = (w_{11} + w_{21})I_{in}$, and the slope is $w_{11} + w_{21}$.

### 5.2.3 Differential structure

The architecture presented in the previous section can only be used for positive input and output currents. In order to get a two or four-quadrant multiplier, it is necessary to introduce a differential structure on the inputs, the outputs or both.

The differential structure consists of recentering the currents around a reference value, $I_{ref}$, such that all the information is now carried by two strictly positive currents. To do so, the two new currents are defined by: $I^+ = I_{ref} + \frac{I}{2}$ and $I^- = I_{ref} - \frac{I}{2}$. When subtracting the two, the current of interest is recovered: $I^+ - I^- = I$.

In the case of one single multiplier, $b = wa$, with differential input and output, this corresponds to performing the following operation: $\begin{bmatrix} b^+ \\ b^- \end{bmatrix} = \begin{bmatrix} w^+ & w^- \\ w^- & w^+ \end{bmatrix} \begin{bmatrix} a^+ \\ a^- \end{bmatrix}$. To maintain the symmetry, it is usually accepted that:

$$w^+ = |w| + \frac{w}{2}, w^- = |w| - \frac{w}{2}$$

The resulting output is:

$$b = b^+ - b^- = (w^+a^+ + w^-a^-) - (w^-a^+ + w^+a^-) = w(a^+ - a^-) = wa$$

This is achieved by the circuit shown in Figure 29.

Figure 30 shows experimental results of this structure. The positive input was swept linearly from $0nA$ to $10nA$, while the negative input was swept linearly from $10nA$ to $0nA$.

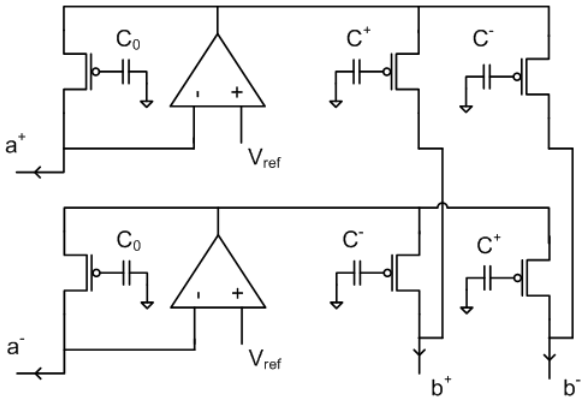


**Figure 29. 1x1 Differential VMM. Two inputs are used to represent the differential input, and two outputs to represent the differential output. Four floating-gate transistors are needed to implement the weight. This block can perform four-quadrant operations.**

The resulting differential input is a linear sweep from $-10nA$ to $10nA$. The transistors' floating-gates were programmed several times to show all of the four-quadrant possibilities for different resulting weights. Only the final output ($I_{out} = b^+ - b^-$) is plotted.

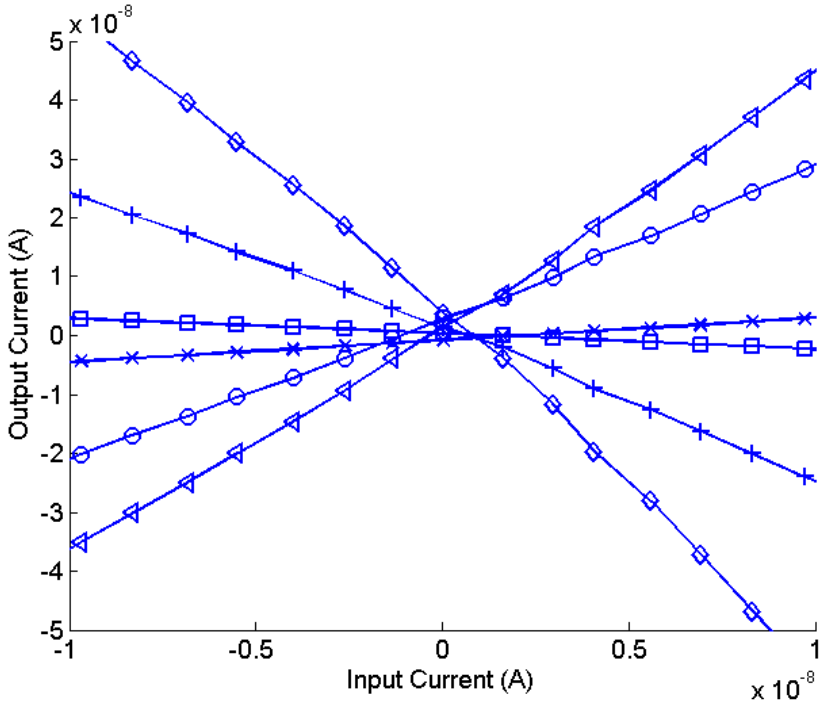


**Figure 30. Experimental results of the 1x1 Differential VMM. The weight was programmed to 6 different values to illustrate the four-quadrant capacity of this multiplier.**

## 5.3 Individual Parts

In this section, the circuit design of each part presented in Chapter 4 will be shown and the results obtained on the FPAA for these blocks will be presented.

### 5.3.1 Inputs

In the input stage of the global system, to convert a voltage into a current, the structure shown in Figure 31 is used. This circuit consists of a floating-gate OTA which performs the conversion from input voltage to output current, according to the equation:

$$I_{out} = g_m(V_{in}^{+} - V_{in}^{-})$$

where $g_m$ is the gain of the amplifier or transconductance.

For this circuit, the negative input of the OTA is connected to Vdd and the positive input voltage can be easily set on the testing board if it is connected to a DAC. With this structure, it is possible to get a positive linear relationship between the input voltage and the output current.

By changing the gain of the OTA, it is possible to vary the range of the output current. The values to which the input floating-gates are programmed allow the output current to move along the x-axis.

Figure 32 shows the experimental result of this circuit obtained on the FPAA.

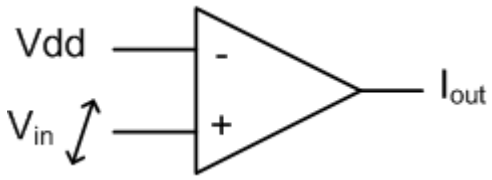


**Figure 31. Implementation of the input stage. This block converts an input voltage to a current.**

It is possible to get a negative relationship between the input voltage and the output current by simply reversing the two inputs: connect Vdd to the positive input of the OTA and set the voltage on the negative input, as shown in Figure 33.

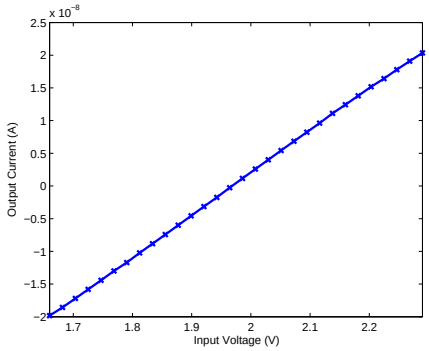


**Figure 32. Experimental results of the input stage. The input voltage to output current relationship is relatively linear. The value of the bias current of the OTA determines the output range.**

Figure 34 shows the experimental result for this structure. As expected, the slope is negative.

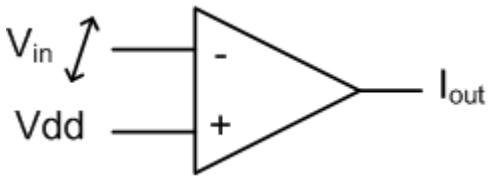


**Figure 33. Implementation of the input stage for a negative input. The output current is negatively related to the input voltage.**

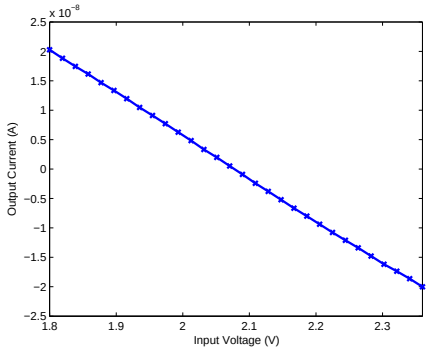


**Figure 34. Experimental Results for a Negative Input. By inverting the two input connections, it is possible to obtain a negative input to output relationship.**

In order to eliminate the dependency on the input characteristics and to be able to generate any input, an interpolation with a polynomial of degree two was performed on the

characteristic of each input stage. Figure 35 shows the input-to-output characteristic of one of the input stages, and the resulting interpolation with a polynomial of degree two.

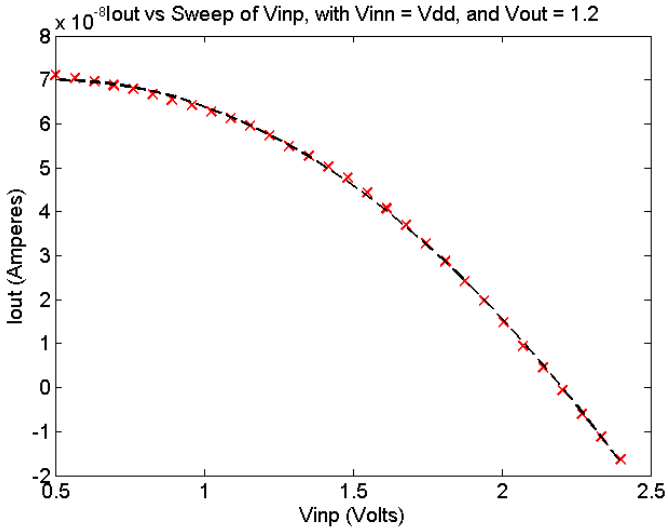


**Figure 35. An interpolation with a polynomial of degree two removed the dependency on the input stage characteristic. The $\times$ symbol are the effective measurements. The dashed line is the interpolation.**

Using this polynomial interpolation, it is possible to generate any desired input. To test the example presented in section 4.3, the two-dimensional input will be swept along the first quarter of the unit circle of radius $20nA$. Two input stages are used to do so, one for each dimension, and the interpolating polynomial is used to determine the values of the input voltages that lead to the desired behavior. The resulting output currents are:

$I_{in1} = cos(\theta) \cdot 20nA$

$I_{in2} = sin(\theta) \cdot 20nA$

For $\theta = [0, \pi/2]$.

The experimental results are shown on Figure 36.

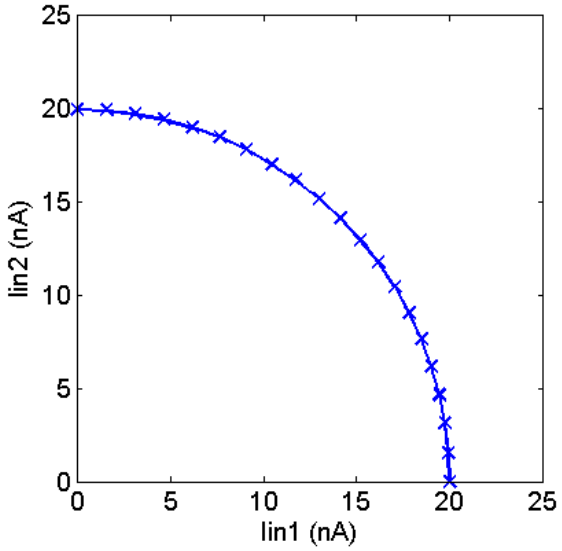


**Figure 36. The two-dimensional input is generated using two input stages. The polynomial interpolation is used to set the voltages which lead to the two currents $I_{in1} = cos(\theta) \cdot 20nA$ and $I_{in2} = sin(\theta) \cdot 20nA$ for $\theta = [0, \pi/2]$. The resulting input is swept along the first quarter of the unit circle.**

### 5.3.2 Outputs

In the output stage, the goal is to transform the current into a voltage. To do so, the circuit shown in Figure 37 uses two floating-gate OTAs. This circuit is known as a transimpedance amplifier (TIA). The top-most OTA acts as a resistance $\frac{1}{g_m}$ which feeds back to the negative input of the second OTA. The resulting output voltage is a linear transform of the input current

$$V_{out} \approx V_{ref} - \frac{I_{in}}{g_m}$$

. Figure 38 shows the experimental result obtained on the FPAA.

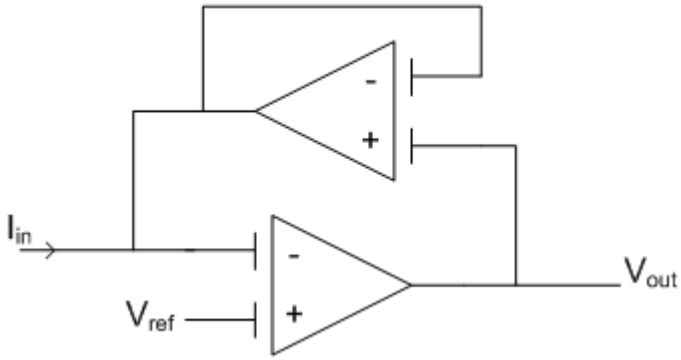


**Figure 37. Implementation of the output stage. The input current is converted to an output voltage to facilitate the measurement on-chip.**

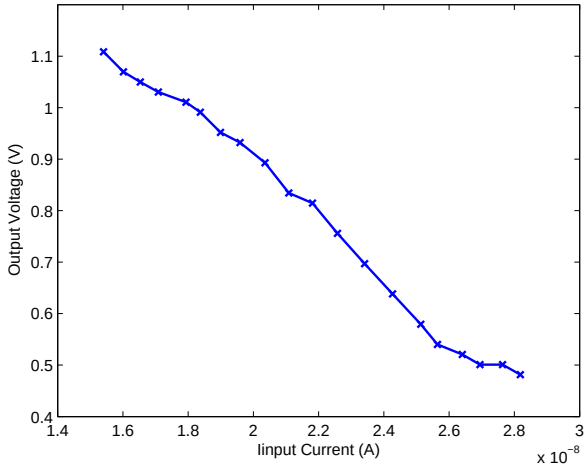


**Figure 38. Experimental result of the output stage. The input-to-output relationship is almost linear.**

### 5.3.3 Driving VMM

As was explained in section 4.2.3, the Driving VMM performs the computation of the driving inputs. Since the inputs and outputs of this block can be either positive or negative, a differential-input/differential-output VMM is needed. The weights are chosen so that the outputs compute the inner products between the input and the nodes in the dictionary. In the example of section 4.3, consider only the three positive nodes and limit the input to

positive values. By doing so, the Driving VMM implements the matrix:

$$\Phi^t = \begin{bmatrix} 1 & 0 \\ 0.6 & 0.8 \\ 0 & 1 \end{bmatrix}.$$

Figure 39 shows the three outputs when the input is swept along the circle of radius $20nA$ between the angles $\theta = 0$ rad and $\theta = \frac{\pi}{2}$ rad (see Fig.36). The discrepancy between measured and expected output currents is due to two major factors, which will be described in greater detail in section 6.2. The first is a mismatch between the doping of the floating-gate elements being programmed and used in the circuit. The second is an unstable reference voltage used during programmation.

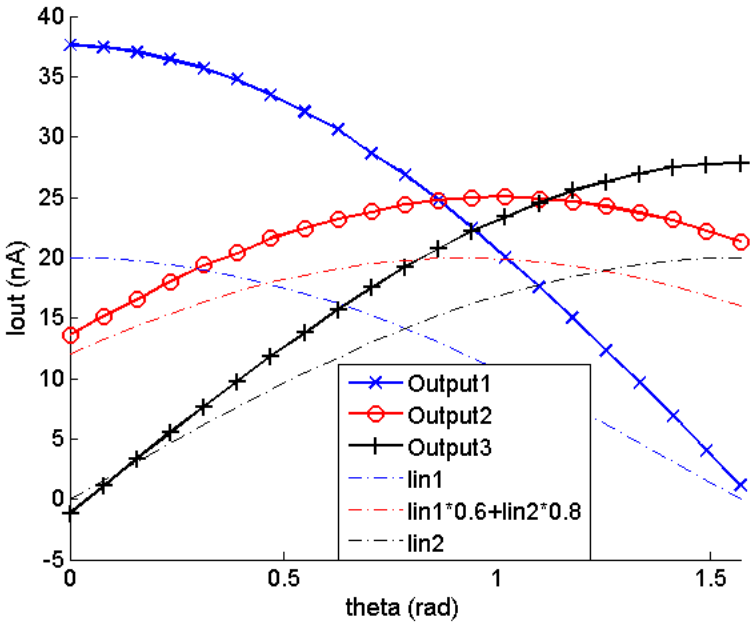


**Figure 39. Experimental results for the DVMM. Both differential current inputs were swept simultaneously along the first quarter of the unit circle. For $\theta = [0, \pi/2]$, $I_{in1} = cos(\theta) \cdot 20nA$, $I_{in2} = sin(\theta) \cdot 20nA$. The dashed lines correspond to the expected outputs. An inconsistent offset in the programmed value of the FGEs caused various scaling issues in the device.**

### 5.3.4 Threshold function

To implement this block, the circuit in Figure 40 is proposed. This circuit performs the operation in (7). In the LCA circuit, there needs to be as many soft-threshold blocks as there are nodes in the system.

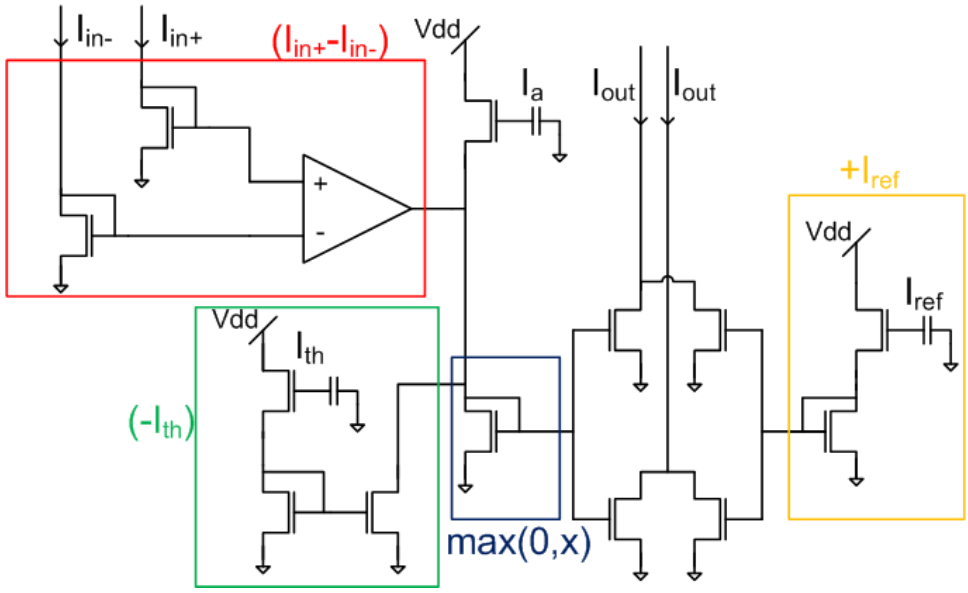


**Figure 40. Analog implementation of the Soft-Threshold with a single sided output. The OTA performs the subtraction of the inputs. The floating-gate $I_{th}$ generates the threshold $\lambda$. The current mirror performs the max operation. The output is copied in order to allow reading and looping back in the circuit.**

Since the input coming from the Driving VMM and the Hopfield VMM are differential, an OTA is used to perform the subtraction of the two differential inputs and recover the current $u = b - h$, as explained in 4.2.4. The floating-gate transistor $I_{th}$ is programmed to generate the desired threshold current. It is passed through a current mirror in order to subtract it from the inputs. The second floating-gate $I_a$ is used in order to tweak the circuit and adjust for possible mismatches in the components. The resulting current is sent through a current mirror in order to perform the thresholding function $(\max(0, x))$ since only positive currents can go through. The last floating-gate transistor $I_{ref}$ adds a bias current to the output, so that the time constant of the Hopfield VMM is reduced. If the input current is negative or below $I_{th}$, no current can go through the current mirror and the

output is zero. Otherwise, the output current is equal to $I_{ref} + I_{in+} - I_{in-} - I_{th} + I_a$. This block was built with two outputs for the purpose of this research. One is looped back into the input of the Hopfield VMM, the other is used to read the global output of the system.

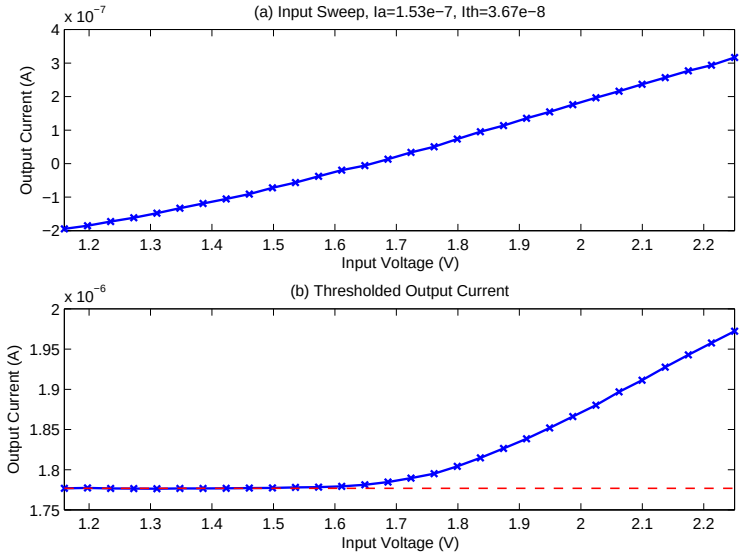


**Figure 41. Experimental results of the Soft-Threshold. The top plot shows the input current, and the bottom plot the resulting output current. As expected, the output is zero bellow a certain threshold and linear with the input above the threshold.**

Figure 41 shows experimental results obtained on the FPAA for this circuit. In this example, the differential input of the soft-threshold was swept linearly. It is plotted on the top graph. The resulting thresholded output current is shown on the bottom graph. As it can be seen, the output is zero below a certain threshold and linear with the input above the threshold.

For the example in section 4.3, the three soft-thresholders were programmed and tested on the FPAA. The FGEs which generate the currents $I_{th}$ and $I_{ref}$ were shared by the three blocks, whereas the FGE which generates $I_a$ was made individual to each block for tunning. The experimental results obtained on the FPAA are shown on Figure 42.

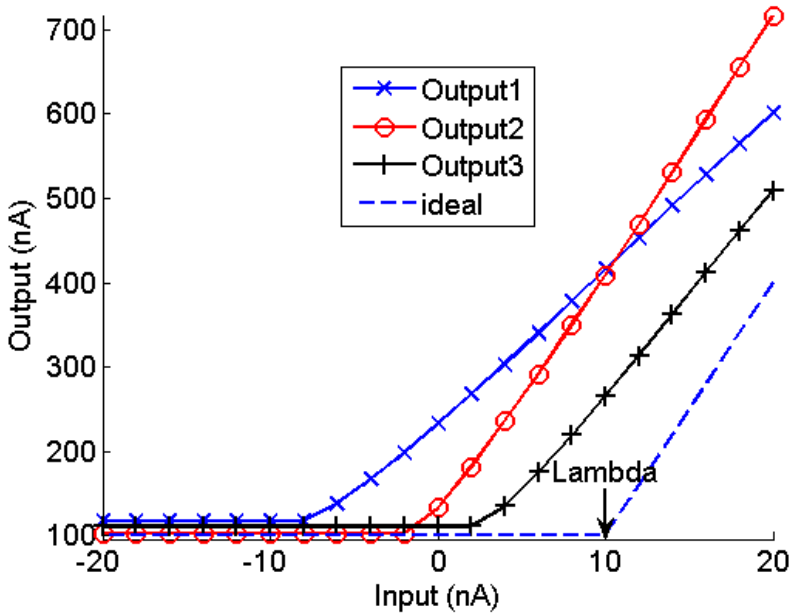


**Figure 42. Experimental results for each soft-threshold node. The three outputs are plotted against a linear sweep of the input current. They are composed of a constant region equal to the reference current for inputs bellow the threshold, and a linear region for inputs above the threshold. The slope can be modified by changing the bias of the OTA, and the threshold value by changing $I_{th}$ and $I_a$.**

### 5.3.5 Hopfield VMM

This block performs the computation of the different feedbacks between the nodes, as explained in section 4.2.5. The inputs of this VMM are single-sided while the outputs are differential. The architecture of this block is shown on Figure 43.

Back to the example in section 4.3, when considering only the positive nodes, the Hopfield VMM has to compute the feedbacks with the following matrix:

$$\Phi^t\Phi - I = \begin{bmatrix} 0 & 0.6 & 0 \\ 0.6 & 0 & 0.8 \\ 0 & 0.8 & 0 \end{bmatrix}$$

The corresponding circuit was programmed on the FPAA and Figure 43 shows the experimental results obtained. Only the input/output relationships leading to non-zero weights are plotted. The input was swept linearly. The dashed line shows the expected

output and the other line is the measured output. As before, the circuit presents the correct behavior but some mismatch can be observed. The two possible causes for this will be presented in section 6.2. Accounting for the mismatch between the transistors improved the results.

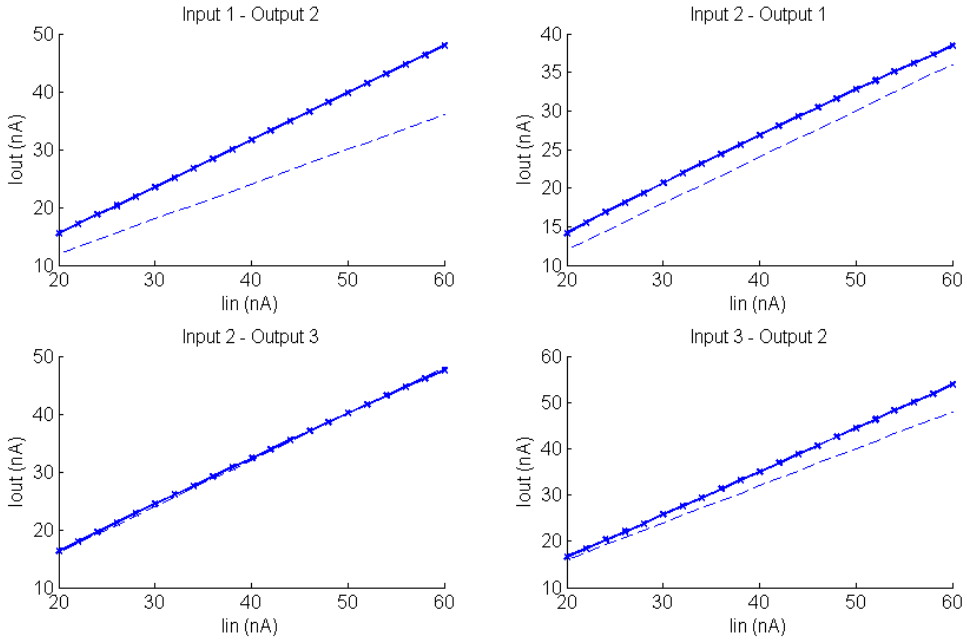


**Figure 43. Experimental results for the Hopfield VMM. Only the input-to-output relationships that led to non-zero outputs are shown. The non-differential input was swept linearly from $20nA$ to $60nA$. The dashed lines show the expected outputs as given by the Hopfield VMM matrix in the example (see section 4.3). Here again, the mismatch issue led to undesired resulting weights that was partly compensated for (see section 6.2).**

# CHAPTER 6

# PROBLEMS ENCOUNTERED AND FUTURE WORK

In the first part of this chapter, the simulation of the entire circuit in WinSpice will be presented. The problems encountered that prevented the testing of the entire system on the FPAA and the next steps that need to be done in this direction will be discussed. Finally, an alternative LCA circuit with an entirely differentiable structure will be presented and a comparison of the two versions will be made.

## 6.1 Spice Simulation of the LCA

Before programming and testing the different parts of the LCA circuit on the FPAA, some simulations were carried out in WinSpice to validate the proposed architecture. Because of the impossibility to test the entire system on the FPAA, the simulation results are presented here to illustrate how the system is expected to work.

Figure 44 shows a simulation of the entire LCA circuit in Spice. In this experiment, the input is of dimension two and there are three nodes of dimension two. They are numbered 1, 2 and 3 and plotted in the graph in the upper left corner. The red arrows represent the two-dimensional input being swept. One dimension of the input, $I_{in}2$, was help constant while the other dimension, $I_{in}1$, was increased linearly. The three remaining graphs represent the three output responses for each node. In red, the dimension of the input which was swept ($I_{in}1$) is plotted for reference. In green, the internal variable $u_i$ is plotted, and in blue, the output of the LCA $a_i$ associated with the corresponding node,$i$ =1 2 or 3, is plotted. As expected, the global outputs $a_i$ are the thresholded versions of the internal variables $u_i$. They are equal to zero whenever the corresponding internal state variable is below the chosen threshold, and linear when the internal variable is above threshold. Moreover, these plots illustrate that the system behaves as expected. At the beginning, when the input vector is closer to the third node, only the output $a_3$ is non-zero. The other two outputs are zero.

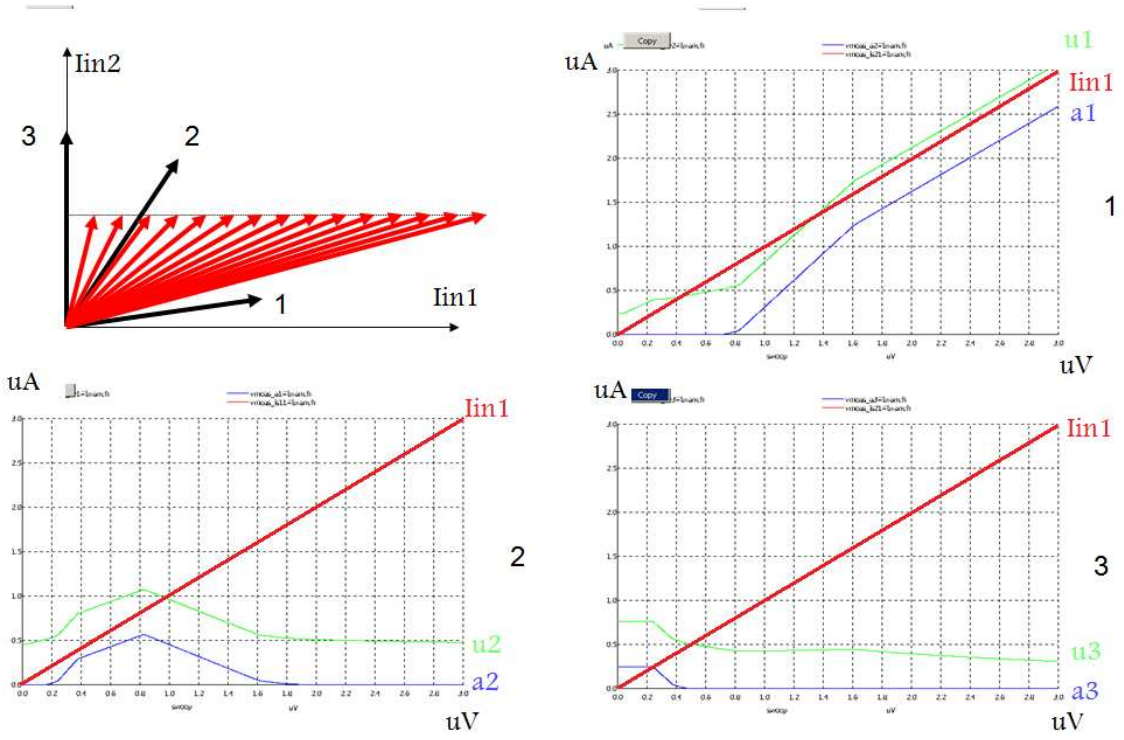


**Figure 44. Simulation of the LCA in Spice. One input was kept constant while the second input was swept linearly. The arrows on the top-left plot represent the 2D-input sweep. In the three graphs, the input current being swept is plotted in red for reference. The internal state variable $u_i$ is plotted in green and the resulting global output $a_i$ is plotted in blue.**

Then, as the input gets closer to node 2, the output associated to node 3 slowly decreases towards zero, while the output associated to node 2 increases. Finally, as the input gets closer to node 1, the output associated to node 2 decreases towards zero, and the output associated to node 1 increases linearly with the input.

Figure 45 [1] shows the result of the simulation of the example described in section 4.3. The global input is swept along the unit circle of radius $20nA$ between the angles $\theta = 0$ rad and $\theta = \frac{\pi}{2}$ rad to enforce a constant energy in input. The LCA circuit outputs are compared to the digital LCA solution.

Figure 46 [1] shows the value of the cost function defined in (3) for the analog LCA solution, compared to the cost for the digital LCA solution and the cost of a trivial orthogonal

[1] Figure courtesy of Sam Shapero.

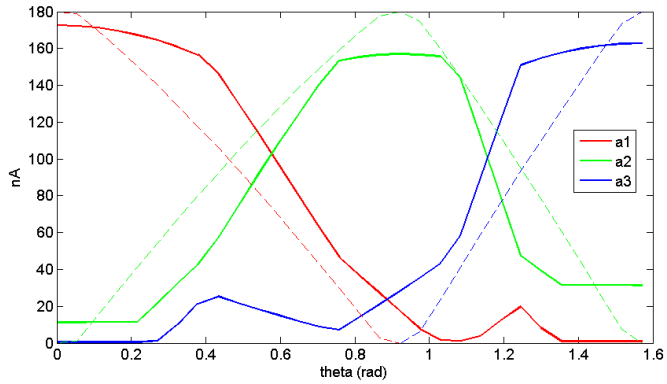


**Figure 45. Simulation of the LCA in Spice. The global inputs were swept along the first quadrant of the unit circle, with $y_1 = cos(\theta) \cdot 200nA, y_2 = sin(\theta) \cdot 200nA$ and $\lambda = 20nA$. The solid a1, a2 and a3 are the output coefficients of the circuit. The dashed lines are a scaled digital solution.**

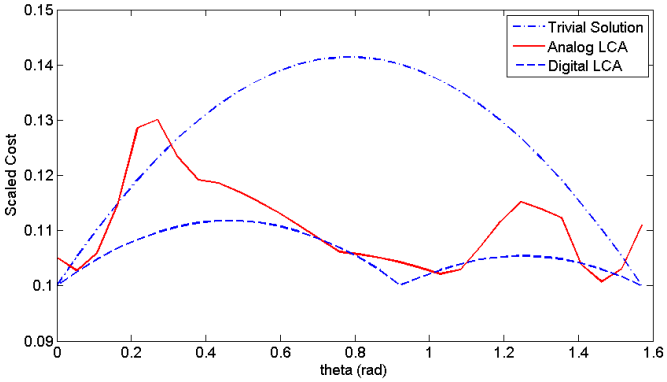


**Figure 46. Comparison of the cost as defined in (3) of the analog LCA solution, the digital LCA solution and a trivial orthogonal projection solution onto node 1 and 3.**

projection onto nodes 1 and 3. To be more precise, these plot show the value:

$$C(a) = \frac{1}{2} \|s - \Phi a\|_2^2 + \lambda \|a\|_1$$

for $a$ obtained with the analog LCA, the digital LCA, or a classic projection on nodes 1 and 3. It can be seen that the analog solution generates a cost which is close to the digital LCA and better than the cost of a trivial projection.

## 6.2 Mismatch Problem

The next step in this research consists of testing the entire circuit on the FPAA. However, some issues due to mismatches in the doping of the floating-gate transistors prevented this study until now. This opens the opportunity for new research to be carried out in parallel to this work. I have had the opportunity to be involved in this research as I was working on the LCA analog implementation.

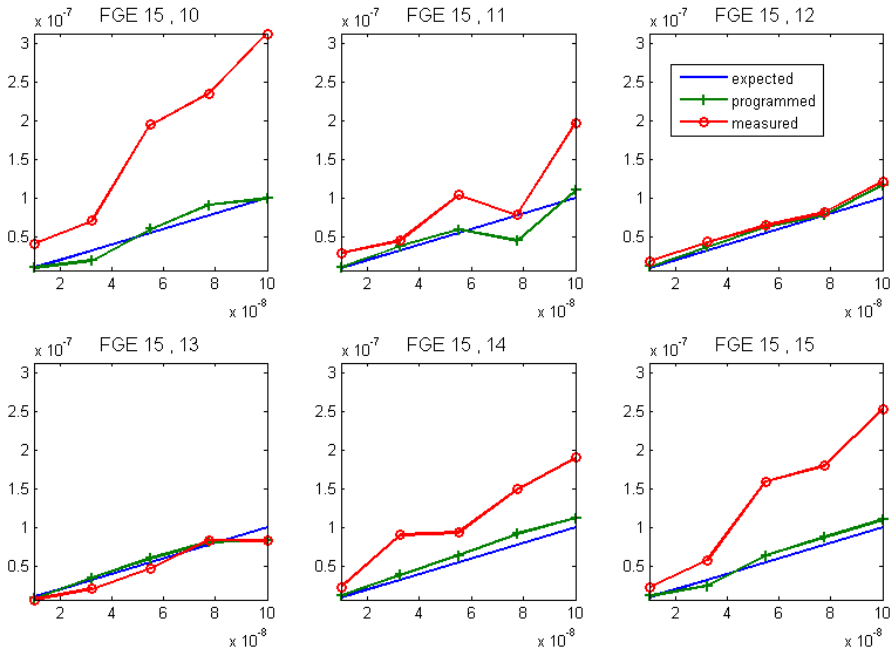


**Figure 47. Expected, programmed and measured current of 6 different FGEs. The FGEs were programmed to 5 target current between** $10nA$ **and** $100nA$**. The errors obtained were up to 50%.**

Figure 47 and 48 show experimental results obtained on the FPAA, which illustrate the mismatch problem. Six floating-gate elements (FGEs) were connected between Vdd and an I/O pin. The I/O pin was connected to an ammeter in order to measure the current going through the FGE. Then, each of the six FGEs were programmed to five different currents between $10nA$ and $100nA$. The current measured on-chip, which is going through the transistor being programmed, will be referred to as the programmed current. The current measured with the ammeter off-chip, which corresponds to the current in the transistor

being used in run mode, will be called the measured current.

On Figure 47, each of the six plots corresponds to one of the six FGEs. The plots contain three curves: the expected current, the programmed current and the measured current. As can be seen, different floating-gate transistors present different behaviors. Some, such as the FGE (10,12), have their programmed and measured currents very close to the expected current, while others, such as the FGE (10,10), have a measured current generally higher than the expected current.

In order to precisely program weights and bias currents in our system, it is necessary to account for this mismatch during the programmation.

We hypothesize that this observed mismatch between programmed and effective currents is due to the indirect programming structure of the FPAA, presented in section 3.2.2.2. The floating-gate transistor, which is being programmed, is not the actual floating-gate transistor used in the circuit (see Fig. 12). The equation of the two currents are the following:

$$I_{meas} = I_{th} \exp \frac{\kappa(V_g - V_{T01})}{U_T}$$
$$I_{prog} = I_{th} \exp \frac{\kappa(V_g - V_{T02})}{U_T}$$

During the fabrication process, some variation in the doping of the transistors leads to a difference in threshold currents $V_{T01} \neq V_{T02}$. As a consequence, the programmed current on the first transistor does not necessary match the actual current in the useful transistor.

However, the ratio of these two currents is:

$$\frac{I_{meas}}{I_{prog}} = \exp \frac{\kappa(V_{T02} - V_{T01})}{U_T}$$

This equation shows that the ratio between the two currents should remain constant. Figure 48 shows the ratio between the measured current and the programmed current for the same six FGEs. The straight line is the average value of this ratio. It can be seen that

the curves obtained are almost constant over the programmed range. However the value of this ratio varies from one FGE to the next. Further work should investigate what other parameters might also intervene in this ratio.

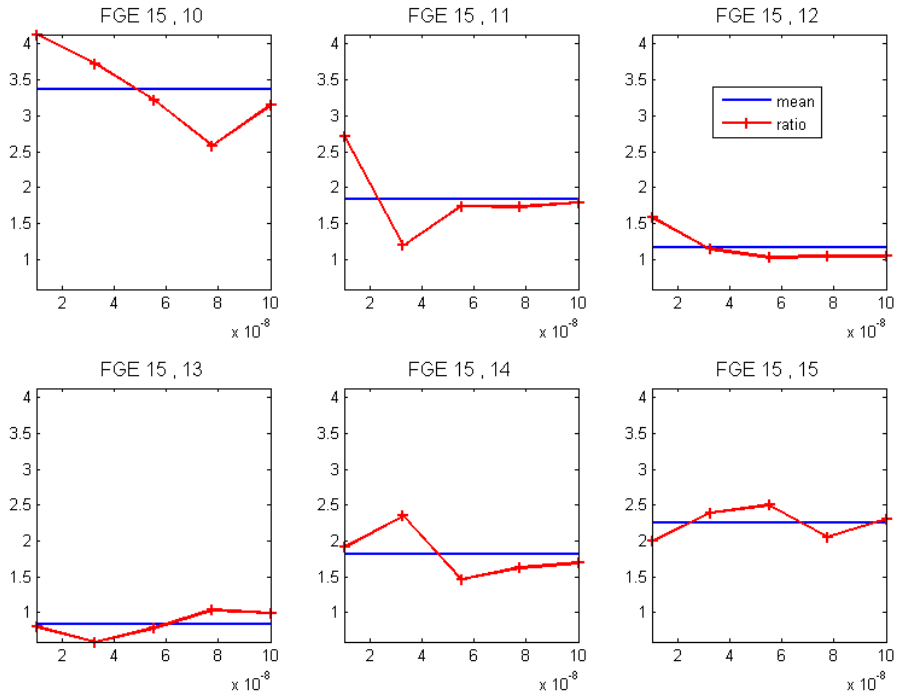


**Figure 48. Ratio between measured and programmed current for 6 different FGEs. The ratio is expected to be a constant. A temperature sensitive reference voltage used in the programming is a possible explanation for the variation of the ratio in those plots.**

In order to find a systematic way to account for this mismatch and compensate for it during the programmation, Sam Shapero and I wrote a Matlab function which automatically generates the list of switches to program on the FPAA in order to measure the on-chip and off-chip currents of the target FGE. This function was used to record the mismatch of the FGEs used in the LCA circuit to generate the different biases and VMM weights. Before each programming, the mismatch recorded was accounted for in the target value of each FGE to program. This helped to reduce the mismatch problem to some extent.

However, another issue led to unsatisfying mismatches between expected and actual

weights in the VMMs. Further research showed that the high temperature sensibility of a reference voltage being used during programming led to these programming errors. Fortunately, the next generation of RASP chip should eliminate this issue by making this reference voltage less temperature dependent.

## 6.3 Alternative LCA circuit

In order to deal with the fact that the current soft-threshold circuit can only deal with positive currents, the current version of the LCA proposes to double the number of nodes in order to get only positive outputs. Another possibility is to build an entirely differentiable soft-threshold circuit. To do so, the soft-threshold block needs to allow the outputs to be differentiable, and implement both halves of the soft-threshold function, shown in Figure 49. Consequently, the Hopfield VMM also needs to become entirely differentiable.

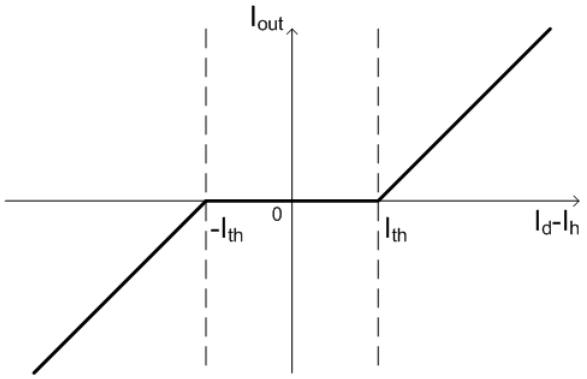


**Figure 49. Plot of the entire Soft-Threshold function. The output is zero when the input is below the threshold in absolute value, and linear with the input otherwise.**

The soft-threshold circuit proposed to implement the entirely differentiable soft-threshold block was designed by Sam Shapero and is drawn on Figure 50.

- If the input current is negative and less than $-I_{th}$. Then, the current going through the top-most OTA is negative and the positive output $I_{out+}$ is zero due to the current mirror. However, the current coming off the bottom-most OTA will be positive. When subtracting the threshold current, the resulting current remains positive and so, the negative output $I_{out-}$

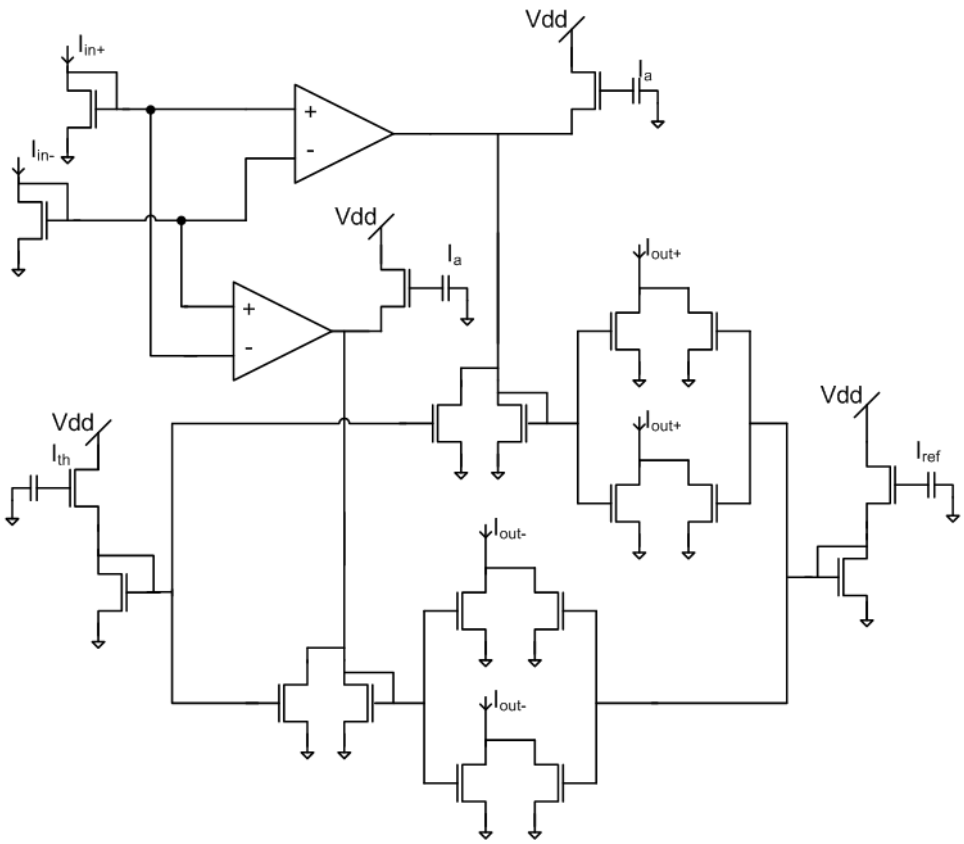


**Figure 50. Analog implementation of the Differential Soft-Threshold. The addition of the second OTA allows to get the negative side of the soft-threshold function.**

is the absolute value of the thresholded input current. When subtracting $I_{out+} - I_{out-}$, the exact thresholded current is recovered.

- In a symmetric way, if the current coming from the second OTA is negative, this results in a negative output $I_{out-}$ being zero, while the output coming from the first OTA is positive and greater than $I_{th}$, resulting in the thresholded value of the input current. When subtracting the two, the desired current is recovered.

- If the input current is between $-I_{th}$ and $I_{th}$, the two outputs $I_{out+}$ and $I_{out-}$ will be zero.

Contrary to what is done with the current version of the LCA, this differential solution prevents one from having to double the number of nodes. However, the differential soft-threshold contains more transistors and more OTAs, and the Hopfield VMM has to be

entirely differentiable, which increases the number of transistors for this block as well.

Table 1 compares the two structures, by looking at the number of transistors and OTAs used in each of them, for a circuit where the input is of dimension $N$ and which contains $M$ nodes.

**Table 1. Comparison of the two versions of the LCA for a $N$ dimensional input and $M$ nodes.**

| | **Current LCA** | | | **Differential LCA** | | |
|---|---|---|---|---|---|---|
| | # FG transistors | # transistors | # OTAs | # FG transistors | # transistors | # OTAs |
| DVMM | $8NM$ | | $2N$ | $4NM$ | | $2N$ |
| Soft-Th | $6M$ | $20M$ | $2M$ | $4M$ | $16M$ | $2M$ |
| HVMM | $8M^2$ | | $2M$ | $4M^2$ | | $4M$ |
| Total | $2M(4M+4N+3)$ | $20M$ | $2N+4M$ | $2M(2M+2N+2)$ | $16M$ | $2N+6M$ |

The differential structure uses more OTAs, but many fewer transistors, and thus saves more space than the current version. Future work should concentrate on implementing and testing this new solution.

# CHAPTER 7
# CONCLUSION

In this thesis, an analog circuit was proposed as a solution to an important class of optimization problem, known as sparse approximation. The results presented showed that individual parts of this system were successfully implemented and tested on a real piece of hardware, the RASP2.8a chip. These results are encouraging and give good hope that an entirely analog solution based on the LCA circuit presented is possible to solve sparse approximation. However, several problems were encountered during this research that prevented the testing of the entire circuit to date. The first issue was a mismatch between the doping of the transistors being programmed on the FPAA. This problem opens a new research challenge that I and other students in the lab have already started to work on. The second problem was due to an unstable reference voltage during programmation. This issue should be resolved in the next generation of RASP chips. Finally a slightly modified version of the circuit, based on the same architecture but composed of entirely differentiable blocks, was presented. In the future, this new system should be tested since it requires fewer transistors.

An analog solution to the sparse approximation problem should provide a significant gain in speed, along with less power consumption and better scaling properties, than can be found in digital solutions. Such a system would have a huge impact on a wide range of applications where this problem arises. One of these fields is image processing. In Magnetic Resonance Imaging, for instance, if one applies the theory of compressed sensing in order to take fewer measurements [24], and then has a fast analog solver to recover the image, it becomes conceivable to perform real-time imaging.

# APPENDIX A

# SOFT-THRESHOLD PARAMETERS

It is important to know how to set the parameters of the soft-thresholder, and in particular the bias current of the OTA. The input stage of the soft-threshold block is plotted on Figure 51.

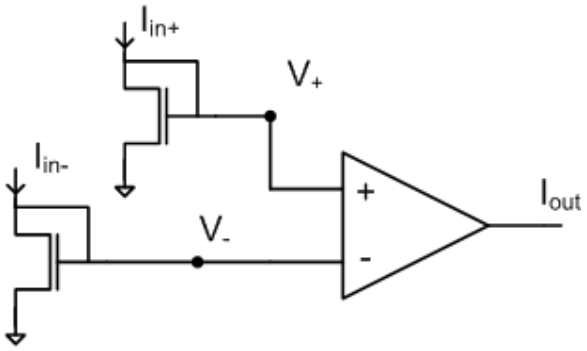


**Figure 51. The input currents are converted to voltages throught the two input transistors. The voltages determine the current in output of the OTA.**

The input currents are converted to voltages by the two transistors in input. The equation that rules these nFet transistors in sub-threshold mode is the following:

$$I = I_0 e^{-\frac{\kappa V_g}{U_T}} \left( e^{\frac{V_s}{U_T}} - e^{\frac{V_d}{U_T}} \right)$$

where $V_g$ is the gate voltage, $V_s$ is the source voltage and $V_d$ is the drain voltage.

For this circuit, the following is true: $V_s = 0$ and $V_d \gg U_T$. As a consequence, the two input currents satisfy:

$$I_{in+} = I_0 e^{\frac{\kappa V_+}{U_T}}$$
$$I_{in-} = I_0 e^{\frac{\kappa V_-}{U_T}}$$

Then the input voltages are:

$$V_+ = \frac{U_T}{\kappa} \ln \frac{I_{in+}}{I_0}$$
$$V_- = \frac{U_T}{\kappa} \ln \frac{I_{in-}}{I_0}$$

The input-to-output characteristic of the OTA is the following:

$$\begin{aligned} I_{out} &= I_{OTA} \tanh\left(\frac{\kappa(V_+ - V_-)}{2U_T}\right) \\ &= I_{OTA} \frac{e^{\frac{\kappa(V_+ - V_-)}{2U_T}} - e^{-\frac{\kappa(V_+ - V_-)}{2U_T}}}{e^{\frac{\kappa(V_+ - V_-)}{2U_T}} + e^{-\frac{\kappa(V_+ - V_-)}{2U_T}}} \end{aligned}$$

From the previous equations and taking into account the bias of the input currents $I_{in+} = I_{bias} + \frac{I_+}{2}$ and $I_{in-} = I_{bias} + \frac{I_-}{2}$, the voltage is:

$$\begin{aligned} V_+ - V_- &= \frac{U_T}{\kappa}\left(\ln I_{bias} + \frac{I_+}{2} - \ln(I_0) - \ln I_{bias} + \frac{I_-}{2} + \ln(I_0)\right) \\ &= \frac{U_T}{\kappa} \ln\left(\frac{I_{bias} + \frac{I_+}{2}}{I_{bias} + \frac{I_-}{2}}\right) \end{aligned}$$

Plugging this expression back in the expression of the output current gives:

$$\begin{aligned} I_{out} &= I_{OTA} \frac{e^{\frac{1}{2}\ln(\frac{I_{bias}+\frac{I_+}{2}}{I_{bias}+\frac{I_-}{2}})} - e^{-\frac{1}{2}\ln(\frac{I_{bias}+\frac{I_+}{2}}{I_{bias}+\frac{I_-}{2}})}}{e^{\frac{1}{2}\ln(\frac{I_{bias}+\frac{I_+}{2}}{I_{bias}+\frac{I_-}{2}})} + e^{-\frac{1}{2}\ln(\frac{I_{bias}+\frac{I_+}{2}}{I_{bias}+\frac{I_-}{2}})}} \\ &= I_{OTA} \frac{\sqrt{\frac{I_{bias}+\frac{I_+}{2}}{I_{bias}+\frac{I_-}{2}}} - \sqrt{\frac{I_{bias}+\frac{I_-}{2}}{I_{bias}+\frac{I_+}{2}}}}{\sqrt{\frac{I_{bias}+\frac{I_+}{2}}{I_{bias}+\frac{I_-}{2}}} + \sqrt{\frac{I_{bias}+\frac{I_-}{2}}{I_{bias}+\frac{I_+}{2}}}} \\ &= I_{OTA} \frac{\sqrt{I_{bias}+\frac{I_+}{2}^2} - \sqrt{I_{bias}+\frac{I_-}{2}^2}}{\sqrt{I_{bias}+\frac{I_+}{2}^2} + \sqrt{I_{bias}+\frac{I_-}{2}^2}} \\ &= I_{OTA} \frac{\frac{I_+ - I_-}{2}}{2I_{bias} + \frac{I_+ - I_-}{2}} \\ &\approx \frac{I_{OTA}}{4I_{bias}}(I_+ - I_-) \qquad \text{because } I_+, I_- \ll I_{bias} \end{aligned}$$

This generalizes for more than two inputs. As a consequence, the bias of the OTA has to scale with $I_{bias}$ and the number of inputs, so that the gain $\frac{I_{OTA}}{4MI_{bias}}$ equals one.

# REFERENCES


[1] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM review*, vol. 43, no. 1, pp. 129–159, 2001.

[2] G. Frantz, "Digital signal processor trends," *IEEE Micro*, vol. 20, pp. 52–59, 2000.

[3] C. M. Twigg, P. Hasler, and D. Anderson, "Large-scale FPAA devices for signal processing applications," *IEEE Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. II–69–72, 2007.

[4] B. Natarajan, "Sparse approximate solutions to linear systems," *SIAM Journal on Computing*, vol. 24, no. 2, pp. 227–234, 1995.

[5] D. Donoho and M. Elad, "Optimally sparse representation in general (non-orthogonal) dictionaries via L1 minimization," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 100, no. 5, pp. 2197–2202, 2003.

[6] E. Candes, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, pp. 1207–1223, Aug. 2006.

[7] E. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. on Information Theory*, vol. 52, pp. 489–509, Feb. 2006.

[8] D. Donoho, "Denoising by soft-thresholding," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 613–627, 1995.

[9] C. Rozell, D. Johnson, R. Baraniuk, and B. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, pp. 2526–2563, Oct. 2008.

[10] J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci. USA*, vol. 81, no. 10, pp. 3088–3092, 1984.

[11] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Developing large-scale field-programmable analog arrays," *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 142, Apr. 2004.

[12] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "Rasp 2.8: A new generation of floating-gate based field programmable analog array," *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pp. 213–216, Sept. 2008.

[13] D. Kahng and S. Sze, "A floating-gate and its application to memory devices," *The Bell System Technical Journal*, vol. 46, no. 4, pp. 1288–1295, 1967.

[14] P. D. Smith, M. Kucic, and P. Hasler, "Accurate programming of analog floating-gate arrays," *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 5, pp. 489–492, May 2002.

[15] D. Abramson, "MITE architectures for recongurable analog arrays," Master's thesis, Georgia Institute of Technology, 2004.

[16] C. Duffy and P. Hasler, "Modeling hot-electron injection in pFETs," *Journal of Computational Electronics 2*, pp. 317–322, 2003.

[17] C. M. Twigg, J. D. Gray, and P. E. Hasler, "Programmable floating gate FPAA switches are not dead weight," *IEEE Proceedings of the International Symposium on Circuits and Systems*, pp. 169–172, 2007.

[18] D. W. Graham, E. Farquhar, B. Degnan, C. Gordon, and P. Hasler, "Indirect programming of floating-gate transistors," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, pp. 951–963, May 2007.

[19] C. R. Schlottmann, "Analog signal processing on a reconfigurable platform," Master's thesis, Georgia Institute of Technology, 2009.

[20] C. Petre, C. Schlottmann, and P. Hasler, "Automated conversion of simulink designs to analog hardware on an FPAA," *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 500–503, May 2008.

[21] F. Baskaya, S. Reddy, S. K. Lim, and D. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, pp. 906–910, Aug. 2006.

[22] F. Baskaya, B. Gestner, C. Twigg, S. K. Lim, D. Anderson, and P. Hasler, "Rapid prototyping of large-scale analog circuits with field programmable analog array," *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pp. 319–320, Apr. 2007.

[23] F. Baskaya, S. Reddy, S. K. Lim, and D. Anderson, "Hierarchical placement for largescale FPAA," *Field Programmable Logic and Applications, 2005. International Conference on*, pp. 421–426, Aug. 2005.

[24] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic Resonance in Medicine*, vol. 58, pp. 1182–1195, Dec. 2007.